Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Natàlia Gasol Vilamajó

Titulació: Grau en Enginyeria Electrònica Industrial i Automàtica

Títol de Treball Final de Grau: **Sistema de vigilància basat amb sensors PIR i microordinadors**

Director/a: **Ramón Béjar Torres , Carles Mateu Piñol**

Presentació

Mes: Setembre

Any: 2018

Resum

L'objectiu principal d'aquest treball de final de grau és el disseny i la posterior implementació d'un sistema de seguretat amb el microordinador Raspberry Pi mitjançant sensors passius de moviment, capaç de detectar presència no desitjada en un cert espai delimitat i generar avisos de forma remota.

En quan a l'aplicació creada, al llarg del treball s'expliquen les diferents eines i recursos utilitzats, com és, per exemple, els sensors o les diferents llibreries emprades. Es mostra també el servidor web creat per a que l'usuari pugi accedir al registre de tots el moviments detectats pel sistema.

A més, es parla futures ampliacions i millores del sistema que es podrien realitzar en aquest projecte a mode de continuació.

Abstract

The main goal of this end of degree project is the designe and the subseqüent implementation of a security systim with Raspberry Pi microcomputer through passive motion sensors, able to detect the not desired precence in a certain delimited area and notify properly the user by a remote way.

Regarding the application created, throughout the work there are explained the diferent tools and resources used, as it is, for example, the sensors or the diferent libràries. It is also shown the web server created in order to the user could access on the register of all movements detected by the system.

In addition, is talked about future extensions and system improvements that could be made in this project as a continuation.

Índex

Índex de figures.

1. Introducció.

L'avanç de les tecnologies permet dissenyar sistemes de vigilància d'una manera molt senzilla amb elements de molt baix cost, fent tot plegat un tema molt interessant a l'hora d'experimentar amb ell.

L'objectiu principal del present treball és dissenyar i implementar un sistema de detecció de moviment mitjançant sensors de presència connectats a un sistema informàtic que pugi detectar la presència de persones en una zona delimitada i generar avisos adequats quan es detecti moviment.

A més, un dels altres objectius que tenim és que aquest sistema que implementem sigui de baix cost, per a que es pugui utilitzar en diversos entorns sense haver de fer una despesa important de diners. Així doncs, els sensors i microordinadors utilitzats hauran de ser de baix cost.

Tot i haver molts microordinadors disponibles per a poder implementar el sistema, cal dir que en nostre cas, un dels requeriments es que sigui amb la Raspberry Pi. Principalment ens hem centrat en aquest microordinador degut al gran suport amb el compte.

Un dels aspectes que tractarem en el nostre projecte és el consum energètic, ja que es un dels requeriments principals a l'hora d'implementar sistemes de monitorització els quals han de funcionar les 24 hores del dia, que el seu consum sigui el més baix possible.

També cal dir que, un altre dels requeriments d'aquest projecte és que el sistema avisi de forma remota als usuaris sobre les alertes dels diferents esdeveniments de presencia que s'han produït durant el seu funcionament.

2. Disseny del sistema.

Com ja hem comentat en la introducció el nostre microordinador escollit és el anomenat Raspberry Pi.

Es consideraran diversos escenaris a l'hora d'utilitzar els sensors de moviment, depenent de la grandària de l'espai on s'implanti aquest sistema.

- En habitacions de dimensions reduïdes, es farà la implementació d'una solució en la qual hi ha diversos sensors connectats a la Raspberry Pi.
- Per espais més grans, és farà el disseny d'una altra situació en la qual els diferents sensors estan connectats cadascun a un microcontrolador ESP de manera individual i s'enviaran les dades a traves d'una xarxa local Wifi a una Raspberry Pi central.

Un dels elements principals del nostre sistema són els sensors de presència que generen un senyal alt o baix en funció de si han detectat moviment o no. Hem escollit sensors de baix cost i també de baix consum elèctric, factor molt important, com ja hem dit, en sistemes de monitorització que han d'estar actius 24 hores al dia.

Un altre mòdul del sistema és el programa principal de monitoratge, que estarà ubicat en el microordinador, que en el nostre cas és la Raspberry Pi. A més, també hi haurà una base de dades ubicada en el mateix ordenador que permetrà emmagatzemar els diferents esdeveniments detectats.

Aquesta base de dades podria estar ubicada en un altre microordinador, però per raons de comoditat em decidit instal·lar-la en el mateix que conté el programa principal.

S'ha implementat també un servidor web que permetrà als usuaris que realitzar un seguit de qüestions al sistema sobre els esdeveniments detectats. Les qüestions que podrà realitzar són les següents:

- Podrà consultar els moviments detectats al llarg de tot el funcionament del sistema de seguretat que hagin transcorregut, òbviament, utilitzant el mateix microordinador.
- Podrà seleccionar el període temporal que desitja veure o comprovar, introduint les dates que vulgui i el sistema li mostrarà els esdeveniments detectats en aquell període.

A més, cada cop que un moviment sigui detectat s'enviarà una alerta. En el nostre projecte, hem escollit el servidor SMTP per enviar les dites altertes ja que és el estàndard de Internet més utilitzat a dia d'avui.

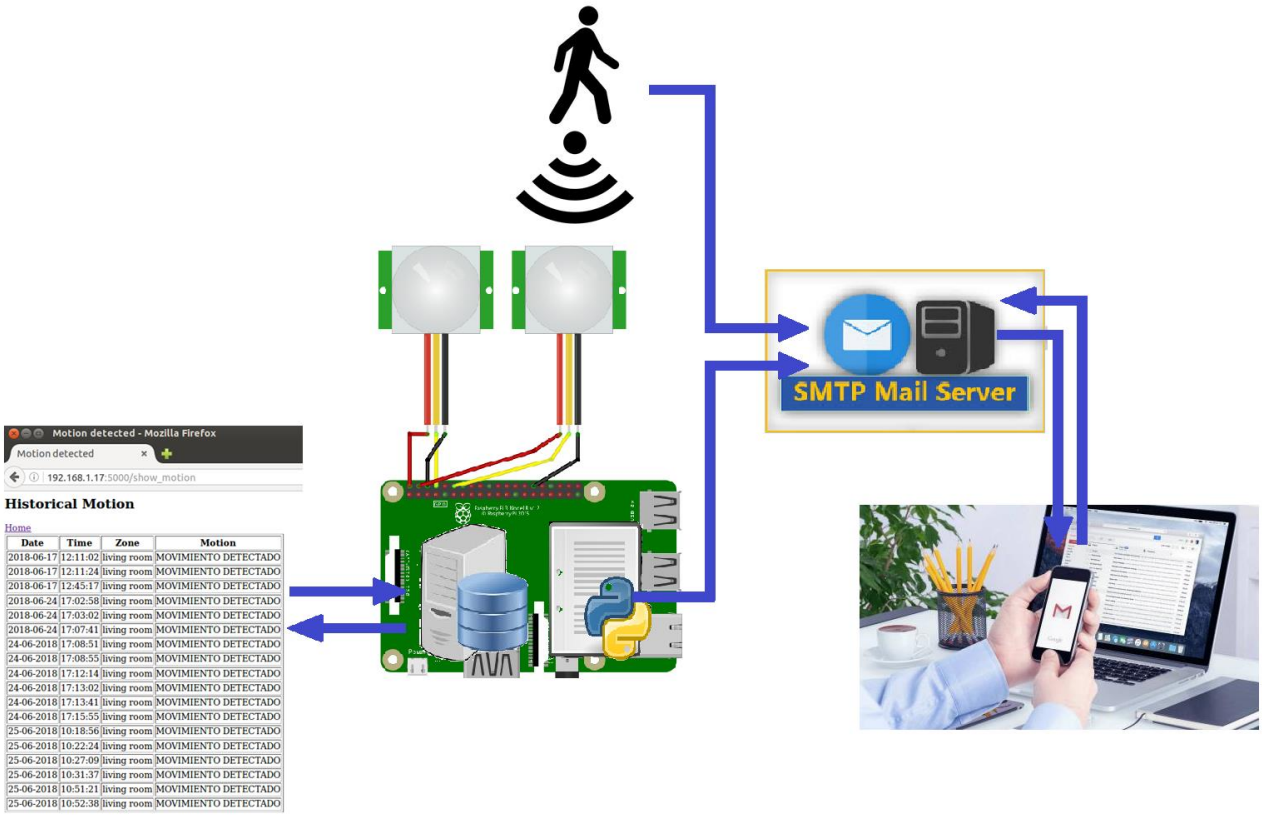
Les alertes seran enviades de diferent manera segons l'usuari elegeixi.

- El primer mode enviarà les alertes cada cop que un esdeveniment sigui detectat.
- El segon mode el que farà serà agrupar alertes que ocorrin en un lapse de temps suficientment petit com per a ser considerats un mateix esdeveniment i llavors, informarà a l'usuari indicant quants moviments i en quant temps han tingut lloc.

També és compta amb un arxiu de configuració on l'usuari ha d'introduir-hi certes dades que dependran de cada muntatge del sistema i d'elles en depèn el correcte funcionament d'aquest.

En aquest fitxer de configuració l'usuari també haurà d'elegir en quin mode vol ser notificat segons les seves preferències.

A continuació es mostra un petit esquema del que seria el disseny pensat per al nostre projecte, resumint de manera visual el que anteriorment acabem d'explicar.



Date	Time	Zone	Motion
2018-06-17	12:11:02	living room	MOVIMIENTO DETECTADO
2018-06-17	12:11:24	living room	MOVIMIENTO DETECTADO
2018-06-17	12:45:17	living room	MOVIMIENTO DETECTADO
2018-06-24	17:02:58	living room	MOVIMIENTO DETECTADO
2018-06-24	17:03:02	living room	MOVIMIENTO DETECTADO
2018-06-24	17:07:41	living room	MOVIMIENTO DETECTADO
24-06-2018	17:08:51	living room	MOVIMIENTO DETECTADO
24-06-2018	17:08:55	living room	MOVIMIENTO DETECTADO
24-06-2018	17:12:14	living room	MOVIMIENTO DETECTADO
24-06-2018	17:13:02	living room	MOVIMIENTO DETECTADO
24-06-2018	17:13:41	living room	MOVIMIENTO DETECTADO
24-06-2018	17:15:55	living room	MOVIMIENTO DETECTADO
25-06-2018	10:18:56	living room	MOVIMIENTO DETECTADO
25-06-2018	10:22:24	living room	MOVIMIENTO DETECTADO
25-06-2018	10:27:09	living room	MOVIMIENTO DETECTADO
25-06-2018	10:31:37	living room	MOVIMIENTO DETECTADO
25-06-2018	10:51:21	living room	MOVIMIENTO DETECTADO
25-06-2018	10:52:38	living room	MOVIMIENTO DETECTADO

Il·lustració 1. Esquema del disseny del projecte.

3. Implementació de la solució dissenyada i tecnologies utilitzades.

Degut a l'objectiu proposat en aquest treball, s'han de començar estudiant diferents maneres d'utilitzar els sensors de moviment mitjançant una Raspberry Pi. A continuació es parlarà de diferents formes possibles de comunicació així com també de possibles sensors de presencia. A més, es parlarà de les llibreries emprades en el llenguatge de programació utilitzat, que en el nostre cas l'escollit ha estat Python.

3.1 Protocols de comunicació.

En la Raspberry Pi hi ha diferents protocols de comunicació:

UART.

Són les sigles en anglès de Universal Asynchronous Receiver-Transmitter, és a dir, Transmissor-receptor Asíncron Universal. És el dispositiu que controla els ports i dispositius sèrie. El terme universal significa que la velocitat de transmissió i el format de dades es poden configurar. Com que es asíncron, no necessita enviar senyals de rellotge juntament amb els senyals de dades. Aquest protocol utilitza dues línies de dades: per enviar (Tx) i rebre (Rx) dades. [1]

El UART pren bytes de dades i transmet els bits individuals de forma seqüencial. En la destinació, un segon UART reensambla els bits en bytes complets. La transmissió sèrie de la informació digital (bits) a través d'un cable únic és molt més efectiva pel que fa a cost que la transmissió en paral·lel a través de múltiples cables. [2]

Per defecte, els pins de transmissió i recepció de UART es troben en GPIO 14 i GPIO 15 respectivament, que són pins 8 i 10 en el capçalera GPIO.

I2C.

Es tracta d'un bus molt senzill amb només dos fils, una línia de dades (SDA) i una línia de rellotge (SCL). Es poden realitzar transmissions sèrie bidireccionals de fins a 100 kbit/s en mode estàndard i 400 kbit/s en mode ràpid.

La Raspberry Pi disposa de dos perifèrics per implementar I2C: el BSC (Broadcom Serial Controller) que implementa el mode mestre, i el BSI (Broadcom Serial Interface) que implementa el mode esclau. [3]

En la Raspberry Pi, l'adreça de I²C estàndard és el primer byte enviat pel mestre, encara que els primers 7 bits representen l'adreça i l'octau bit (R / W-Bit) és el que comunica al esclau si ha de rebre dades del mestres (low / baix) o enviar dades al mestre (high / alt). Per tant, I²C utilitza un espai d'adreça de 7 bits, que permet fins a 112 nodes en un bus (16 de les 128 adreces possibles estan reservades per a fins especials). [4]

Els pins de comunicació I2C es troben en GPIO 0, GPIO 1, GPIO 2 i GPIO 3.

SPI.

El bus d'interfície de perifèrics sèrie o bus SPI, Serial Peripheral Interface, és un estàndard per controlar gairebé qualsevol dispositiu electrònic digital que accepti un flux de bits sèrie regulat per un rellotge (comunicació síncrona). [5]

SPI utilitza 4 connexions separades per comunicar-se amb el dispositiu de destinació. Aquestes connexions són: el rellotge de sèrie (SCLK), Master Input Slave Output (MISO), Master Output Slave Input (MOSI) i la selecció de xip (CS).
Es un protocol de comunicació bastant utilitzat a l'hora de comunicar dades entre microcontroladors com la Raspberry Pi i dispositius perifèrics.

Utilitza 11 pins GPIO: GPIO 11 i 21 actuant com a SCLK, GPIO 9 i 19 com a MISO, GPIO 10 i 20 fan de MOSI i els GPIO 7, 8, 16, 17, 18. [6]

Fent una petita comparació entre el protocol I2C i SPI:

Amb SPI s'obté major velocitat de transmissió que amb I²C a la vegada que es consumeix menys energia, degut a que posseeix menys circuits (incloent les resistències pull-up) i aquests són més simples. La seva implementació en hardware és extremadament senzilla. [7]

També cal dir que, en SPI els dispositius clients usen el rellotge que envia el servidor, no necessiten per tant el seu propi rellotge. No és obligatori implementar un transceptor (emissor i receptor), un dispositiu connectat pot configurar-se perquè només enviï, només rebi o les dues coses a la vegada.

Algunes de les desavantatges del protocol SPI es que consumeix més pins de cada chip el que protocol I2C i que a més, no permet tenir fàcilment diferents servidors connectats al bus.

SPI només pot recórrer distàncies curtes i rarament fora de la PCB mentre que I2C pot transmetre dades a distàncies molt més grans, tot i que a baixes taxes de dades.

La manca d'un estàndard formal ha donat lloc a diverses variacions del protocol SPI, variacions que en gran part s'han evitat amb el protocol I2C

En general, SPI és millor per a aplicacions d'alta velocitat i baixa potència, mentre que I2C és millor per a la comunicació amb una gran quantitat de perifèrics i el canvi dinàmic del rol del dispositiu mestre entre els perifèrics de l'autobús I2C. [8]

1-WIRE.

Es un bus de 1 cable que es pot habilitar al GPIO4. Proporciona dades de baixa velocitat, senyalització i alimentació a través d'un sol conductor. Això s'utilitza habitualment per connectar dispositius de sensors de baix cost.

El bus de 1 cable es un bus mestre-esclau simple que es comunica a través de un únic cable de senyal. Els dispositius es comuniquen en el bus enviant la senyal a terra a través d'una sortida de drenatge obert i mostrejant el nivell lògic de la línia de senyal. El subsistema w1 proporciona el marc per administrar els metres w1 i la comunicació amb els esclaus. Tots els dispositius esclaus w1 han d'estar connectats a un dispositiu mestre d'aquest mateix bus. [9]

3.2 Sensor de moviment.

És cert que hi ha sensor de detecció de presencia actius, que són els que injecten llum, microones o so en el medi ambient i detecten si existeix algun canvi en ell, o bé el passius, els quals no emeten o irradien cap energia per a detectar cap canvi, sinó que el que fan es detectar l'energia emesa per altres objectes i alertar quan hi ha un canvi en ella.

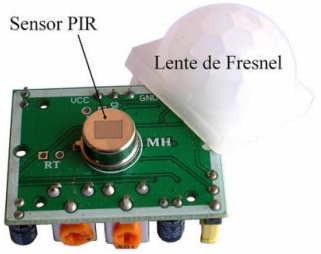
Per al nostre projecte, hem elegit un sensor infraroig passiu, anomenats també PIR (Passive Infrared). Seguidament parlarem del funcionament del sensors PIR així com en concret del sensor HC-SR501. [10]

3.2.1 Principis de funcionament.

La radiació infraroja:
Tots els éssers vius i fins i tot els objectes, emeten radiació electromagnètica infraroja, a causa de la temperatura a la qual es troben. A major temperatura, la radiació augmenta. Aquesta característica ha donat lloc al disseny de sensors d'infrarojos passius, els quals permeten la detecció de moviment, típicament d'éssers humans o d'animals.

Aquests sensors són coneguts com PIR, i prenen el seu nom de 'Pyroelectric Infrared' ó 'Passive Infrared'.

La lent de Fresnel:
La lent de Fresnel és un encapsulat semiesfèric fet de polietilè d'alta densitat, l'objectiu del qual és permetre el pas de la radiació infraroja en el rang dels 8 i 14 micròmetres. La lent detecta radiació en un angle amb obertura de 110 ° i, addicionalment, concentra l'energia en la superfície de detecció del sensor PIR, permetent així una major sensibilitat del dispositiu.



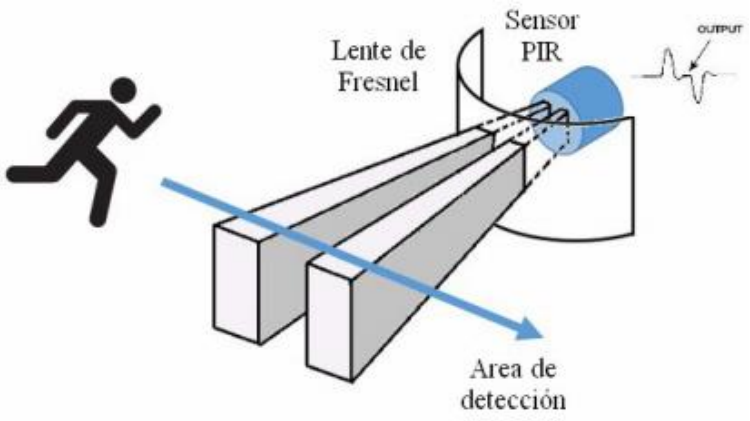
Il·lustració 2. Sensor PIR

El sensor PIR infraroig:

En els sensors de moviment, el sensor PIR consta en realitat de 2 elements detectors separats, sent el senyal diferencial entre aquests el que permet activar l'alarma de moviment. En el cas del HC-SR501, el senyal generat pel sensor ingressa al circuit integrat BISS0001, el qual conté amplificadors operacionals i interfícies electròniques addicionals.

Les funcions i ajustaments complementaris del sensor de moviment són:

- Ajust de paràmetres: mitjançant 2 potenciòmetres, l'usuari pot modificar tant la sensibilitat com la distància de detecció del PIR.
- Detecció automàtica de llum (aquesta funció no està disponible en adquirir el sensor de fàbrica): per mitjà d'una foto resistència CdS (Sulfur de Cadmi), es desactiva l'operació del sensor en cas de que hi hagi suficient llum visible a l'àrea. Aquesta funció és utilitzada en cas de sensors que s'encenguin llums en llocs poc il·luminats durant la nit, i especialment en corredors o escales.

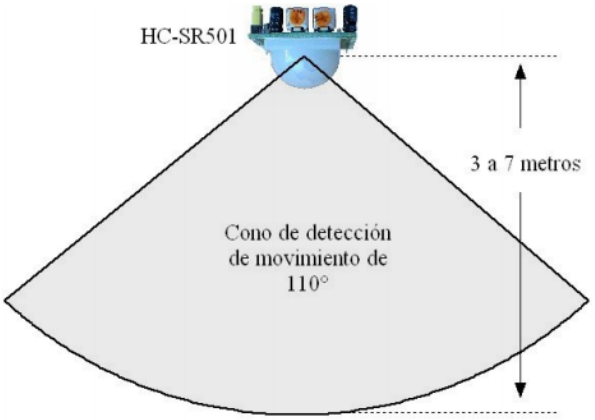


Il·lustració 3. Funcionament del sensor.

Els sensors estan dividits en dues meitats de manera que detectin el canvi de radiació infraroja que reben una i altra banda, disparant l'alarma quan perceben aquest canvi. Així doncs, com es veu en la imatge, s'observa que per a una millor detecció és preferible que la persona travessi paral·lelament per poder detectar el canvi de radiació entre les dues meitats, i no pas que vagi directament cap al sensor.

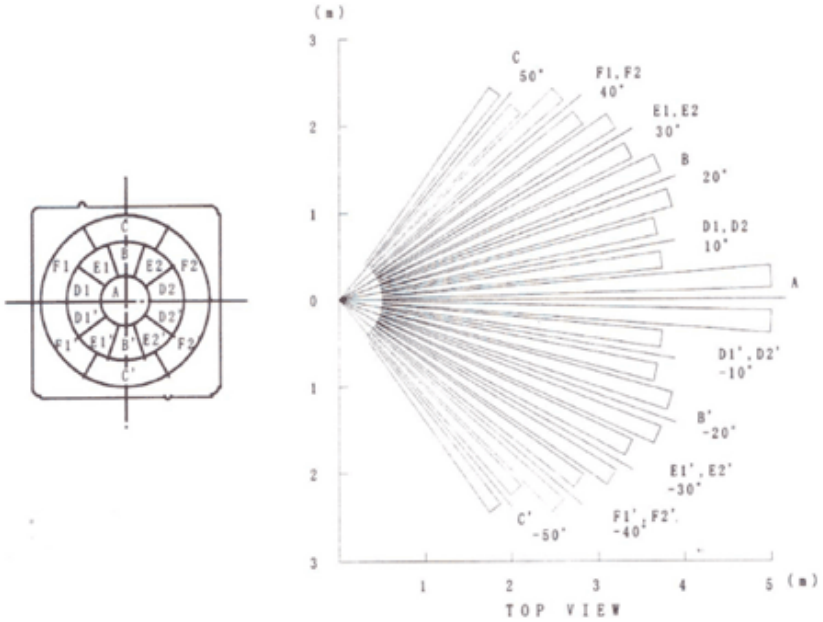
Rang de detecció dels sensors PIR:

Com s'ha indicat anteriorment, el rang de detecció de moviment dels PIR és ajustable i generalment funcionen amb un abast de fins a 7 metres, i amb obertures de 90 ° a 110°, com es mostra a la figura. El muntatge del PIR pot realitzar-se tant en pisos, murs o sostres, segons convingui a l'aplicació.

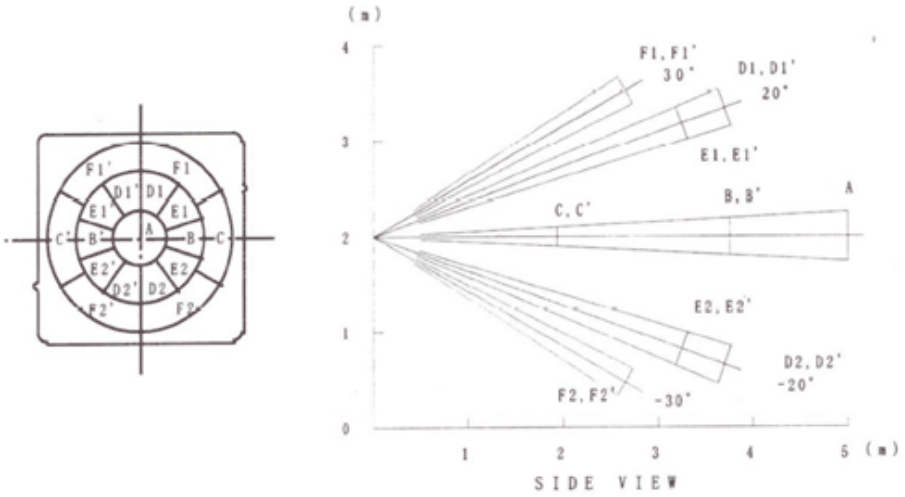


Il·lustració 4. Rang de detecció del sensor HC-SR501.

Cal matisar però, que aquesta obertura de detecció no és igual en les dues direccions, horitzontal i vertical, de l'espai. En les següents imatges es mostra com depenent de la direcció l'angle d'obertura es major o menor. [11]



Il·lustració 5. Vista superior.



Il·lustració 6. Vista lateral.

Horitzontalment s'observa que l'angle d'obertura és major, entre 90° i 110°, però hem d'observar que verticalment aquesta obertura es redueix, obtenint així un angle de detecció entre 60° i 70°.

Aquest factor serà important a l'hora de poder dissenyar la solució desitjada, podem contemplar així tots els punts morts que es quedessin sense detectar pels sensor.

3.2.2 Descripció del HC-SR501.

El mòdul PIR model HC-SR501 és de baix cost, petit, i incorpora la tecnologia més recent en sensors de moviment. El sensor utilitza 2 potenciòmetres i un jumper que permeten modificar els seus paràmetres i adaptar-lo a les necessitats de l'aplicació: sensitivitat de detecció, temps d'activació, i resposta davant deteccions repetitives.

Les seves especificacions tècniques són:

- Utilitza el PIR LHI778 i el controlador BISS0001

- Voltatge d'alimentació: de 5 a 12 VDC

- Consum mitjà: < 1mA

- Rang de distància ajustable de 3 a 7.

- Angle de detecció: con de 110 °

- Ajustaments: 2 potenciòmetres per ajustar tant el rang de detecció com el temps d'alarma activa.

- Jumper per a establir la sortida d'alarma en mode mono-tret o tret repetitiu.

- Sortida d'alarma de moviment amb ajust de temps entre 3 segons fins a 5 minuts.

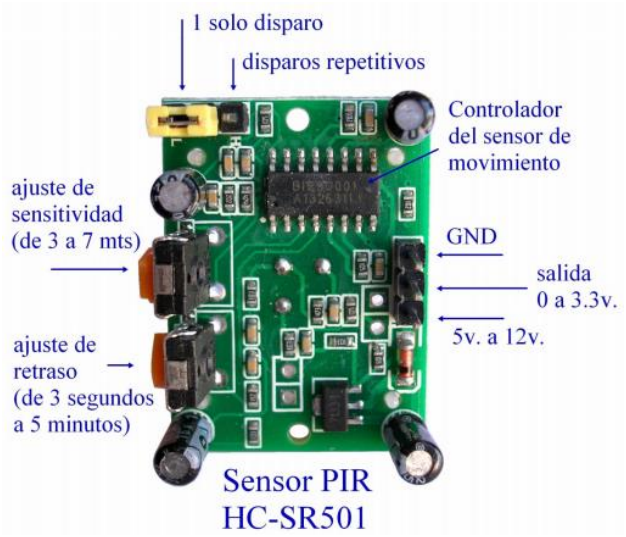
- Sortida d'alarma activa Vo amb nivell alt de 3.3 volts i 5 ma source, a punt per connexió d'un led, o un transistor i relé.

- Temps d'inicialització: després d'alimentar el mòdul HC-SR05, ha de transcórrer 1 minut abans que s'iniciï la seva operació normal. Durant aquest temps, és possible que el mòdul activi 2 o 3 vegades la seva sortida.

- Temps de sortida inactiva: cada vegada que la sortida passi d'activa a inactiva, romandrà en aquest estat els següents 3 segons. Qualsevol esdeveniment que ocorri durant aquest lapse és ignorat.

- Temperatura d'operació: -15 ° a + 70 ° C.

- Dimensions: 3.2 x 2.4 x 1.8 cm

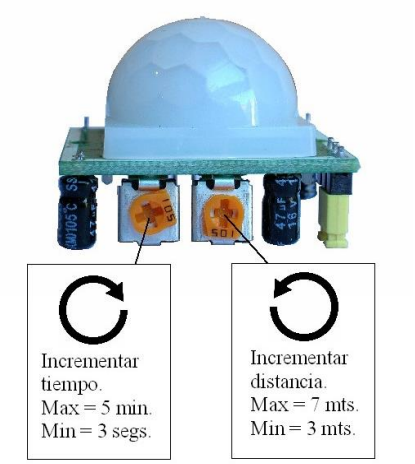


Il·lustració 7. Components sensor PIR HC-SR501

3.2.3 Ajustaments i configuració del sensor.

Potenciòmetres:

D'acord a la figura, l'usuari pot ajustar tant el temps de tret del senyal de alarma de moviment, com la distància de detecció. Els potenciòmetres corresponents han de girar-se en la direcció mostrada per realitzar els ajustos.



Il·lustració 8. Ajustaments sensor PIR HC-SR501

Posició del jumper:

D'acord a la imatge 3 , l'usuari pot treballar en 2 modes d'operació:

- 1 sol tret: en aquest mode, quan hi ha una detecció de moviment (el qual anomenarem 'esdeveniment'), la sortida del sensor s'activa durant el temps que s'hagi ajustat a través del potenciòmetre corresponent. Com exemple, suposem que el temps d'activació és de 60 segons. Si durant aquests 60 segons té lloc un segon esdeveniment, aquest no serà considerat.
- Trets repetitius: en aquest mode, cada esdeveniment detectat genera un nou temps d'activació. Tornant a l'exemple anterior, amb temps d'activació de 60 segons: quan ocorre el primer esdeveniment, la sortida es activa. Si transcorreguts 30 segons passa un segon esdeveniment, llavors es sumaran 60 segons al temps transcorregut, donant un total de 90 segons continus amb la sortida activa. I així, cada esdeveniment addicional, sumarà un temps de 60 segons d'activació al temps ja transcorregut.
- En qualsevol cas, si la sortida torna al seu estat inactiu, hi haurà un lapse de 3 segons durant els quals els nous esdeveniments no seran considerats. Passats aquests 3 segons, el dispositiu torna al seu funcionament normal.

Hi ha sensors en concret que només suporten un tipus de protocol comunicació dels anteriorment mencionats.

En el nostre cas, el sensor PIR adquirit no suporta ni comunicació SPI ni I2C, que potser són dues de les més interessants, en el sentit que o bé transmeten ràpidament o bé permeten connectar una gran quantitat de dispositius al mateix bus, respectivament.

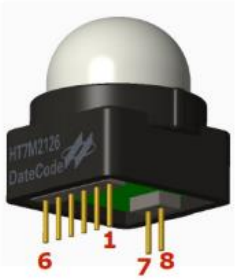
En el cas concret del protocol I2C podríem connectar 112 dispositius i, tenint en compte que la Raspberry Pi disposa de dos busos I2C seriem capaços de tenir-ne fins a 224. Resulta evident que utilitzant els GPIOS seria inviable tenir aquest volum de dispositius i, si be és cert que no necessitarem tants dispositius, podria arribar a ser interessant tenir tres o quatre dispositius en el mateix bus.

Arribats a aquest punt, en quan al protocol de comunicació que s'utilitzarà per comunicar els diferents sensor amb la Raspberry Pi, tenim dues possibilitats. Una es continuar amb el nostre sensor PIR HC-SR501, el qual només podem connectar a través d'un pin GPIO. No seria cap inconvenient ja que com ja he dit, no requerim tanta quantitat de dispositius. L'altra opció es pensar en la possibilitat d'adquirir un sensor PIR que es pugui connectar a través de I2C, bus que considero bastant útil a l'hora de poder connectar diversos dispositius i també interessant ja que no requereix la utilització de molts pins GPIO.

3.3 Sensor PIR HT7M2126.

En veure que el sensor del que disposàvem no acceptava comunicació I2C, vam voler buscar una alternativa a aquest que si ho fes.

Així doncs, vam trobar el HT7M2126, el qual té un consum mitjà d'intensitat <1.5mA quan està en funcionament, i de l'ordre d'uns 50 μA quan està en repòs. [12]



Il·lustració 9. Sensor HT7M2126

El pin MODE/ACT (MODE / DT) s'utilitza per seleccionar el mode xarxa, que ens permetrà utilitzar el bus I2C, o el mode autònom. Quan una resistència que proporciona un estat baix està connectada externament entre aquest pin i el terra, es selecciona el mode autònom. Pel contrari, el mode xarxa es selecciona si una resistència que proporciona un estat alt o cap resistència està externament en aquest pin.

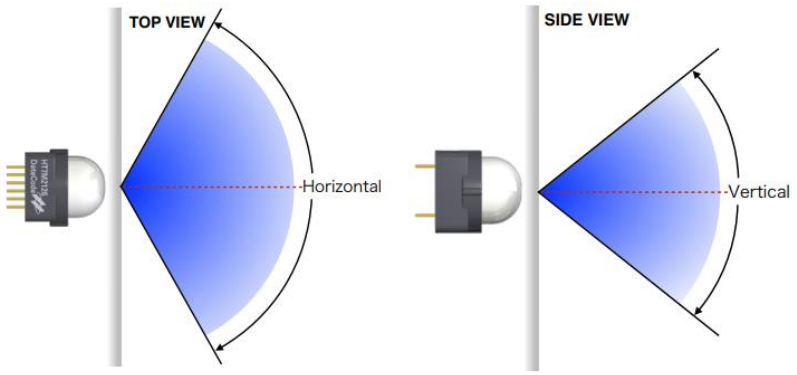
En el nostre cas es interessa treballar en el mode xarxa ja que així ens es possible utilitzar la interfície de comunicació que ens interessa, el I2C.

En aquest mode, l'assignació dels pins es la següent:

PIN	FUNCIÓ	DESCRIPCIÓ
1	VSS	Font de alimentació negativa
2	VDD	Font de alimentació positiva
3	SDA	Entrada/sortida de dades en sèrie pel bus I2C
4	SCL	Línia de rellotge sèrie pel bus I2C (SCL)
5	FTS	Senyal de fototransistor
6	VSS	Font de alimentació positiva
7	MODE/ACT	Selecció de mode / sortida de detecció de moviment
8	TP1	Sense connexió

Angle i distància de detecció

Respecte al sensor HT7M2126, l'angle de visió horitzontal es de 121º, el angle d'inclinació es de 77º i la distància de detecció està entre 3.5 i 6 metres, tal i com es mostra a continuació.



Il·lustració 10. Angle detecció sensor HT7M2126

Té una ràpida estabilització. Està llest per operacions estables en 12 segons després d'haver-lo connectat.

Així doncs, si comparem aquest sensor amb el HC-SR501 veiem que té alguns avantatges, com ara que el temps d'estabilització es molt més ràpid ja que en el HC-SR501 ha de transcorre un minut abans de que inicïi la seva operació amb normalitat. L'altra avantatge principal, motiu pel qual em fer la cerca d'aquest segon sensor és que accepta comunicació I2C, cosa molt interessant si es volen connectar una gran quantitat de sensors.

Tenint en compte que nosaltres no requerim tanta magnitud de sensors, que el sensor HC-SR501 és molt més utilitzat per aplicacions a monitorar amb la Raspberry Pi i que a més, té un consum mitjà lleugerament inferior, decidim quedar-nos amb el sensor que primerament havíem explicat, el HC-SR501.

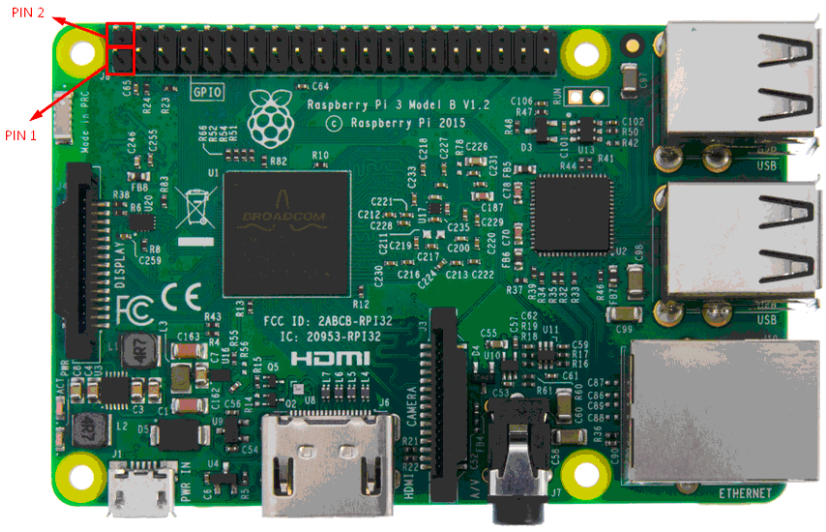
3.4 Llibreries emprades en Python

3.4.1 RPi.GPIO.

Aquest paquet proporciona una classe per controlar els diferents GPIO d'una Raspberry Pi.
A més del control GPIO, també es usat per moltes altres llibreries per consultar la versió hardaware de la Raspberry Pi, des de que els dissenys dels diferents pins canviaven entre certes versions. I, actualment, també ofereix algunes funcionalitats utils de software PWM en tots els pins GPIO. [13]

En el nostre cas l'hem utilitzat dos funcions d'aquest mòdul. Una d'elles es la setmode() i l'altra la setup(). La primera funció nombrada serveix per fe la declaració del numero de pin utilitzat. Els pins estan numerats de dues maneres:

1. GPIO.BOARD - Esquema de numeració de la placa. Especifica que s'està referint als pins pel seu numero, és a dir, als números impresos en la Raspberry Pi. Essent en la imatge els pin 1 i el pin 2 els senyalats, la resta de pins es numeraria d'esquerra a dreta. Per tant, el pin de sota del pin 1 seria el pin 3 i tota aquella fila serien els imparells i l'oposada els parells.



Il·lustració 11. Microordinador Raspberry Pi.

2. GPIO.BCM: números de pin específics del chip Broadcom. Aquests números de pin segueixen un sistema de numeració de nivell inferior definit pel cervell del chip Broadcom de la Raspberry Pi. Els números dels GPIOs, a diferencia del cas anterior, no són correlatius. [14]

En els nostre cas hem utilitzat el mode BOARD degut a la senzillesa que proporciona poder saber el número de pin podent-ho contar físicament.

L'altra funció, la setup(), ens serveix per indicar el numero de pin escollit i per definir el propòsit general d'aquest, és a dir, si volem utilitzar, com a INPUT (per a rebre dades de sensors) o com a OUTPUT (per poder encendre/apagar aparells).

3.4.2 Time.

El mòdul time de la biblioteca estàndard de Python proporciona un conjunt de funcions per treballar amb dates i /o hores. A més d'aquestes funcions, també n'hi ha d'altres relacionades en els mòduls datetime i calendar.

En el nostre cas, per saber el temps (data i hora) en la que s'ha produït la detecció de moviment, la funció localtime() és la utilitzada. El temps s'expressa com un objecte struct_time i ajudant-nos de la funció strftime() es retorna una cadena amb el format que nosaltres elegim.

La funció strftime(format), com ja hem dit, converteix l'estructura de temps que retorna, en el nostre cas, la funció localtime(), tot i que també podria ser la gmtime() a una cadena de text com s'especifica en el format. El format ha de ser obligatòriament una cadena de text. [15]

Aquestes funcions són les mateixes que utilitzem a l'hora de guardar la data a la base de dades.

3.4.3 Sqlite 3.

SQL (Structured Query Language) és un llenguatge de programació de propòsit especial dissenyat per administrar dades emmagatzemades en un sistema de administració de bases de dades relacionals (RDBMS) o per al processament de flux en un sistema d'administració de flux de dades relacionals (RDSMS). [16]

SQLite es una llibreria que implementa un motor de base de dades SQL transaccional, independent, sense servidor i de configuració zero.

El codi per a SQLite es de domini públic i per tant, es gratuït per a qualsevol ús, ja sigui privat o comercial. SQLite és la base de dades més implementada al món amb més aplicacions de les que es podrien arribar a contar, incloent entre elles diversos projectes de alt perfil.

Començant a parlar de la base de dades utilitzada en el nostre projecte, cal dir que, tot i que SQLite3 no es una base de dades amb totes les funcionalitats, n'admet un conjunt bastant elevat del estàndard SQL, i és ideal per aquells que comencen a introduir-se a aprendre SQL. [17]

Així doncs, aquesta llibreria en permet, amb totes les funcionalitats que inclou, crear una base de dades, connectar-nos-hi i guardar-hi les dades que hi desitgem per posteriorment poder-les consultar.

3.4.4 Flask.

Flask és un framework web, és a dir, un entorn de treball web.

Això significa que Flask proporciona eines, llibreries i tecnologies que permeten crear una aplicació web. Proporciona:

- Werkzeug, una biblioteca de utilitats WSGI: com un processador CGI.
- Jinja2 que és el seu motor de templates: serà útil quan és faci HTML.
- Flask incorpora un petit servidor web (només per a propòsits de desenvolupament)

Amb les templates (plantilles) podem definir pàgines generals i deixar algunes parts depenent de les variables. [18]

En el nostre projecte ha estat utilitzat per poder fer l'aplicació web que ens permet consultar els moviments enregistrats i permetre així que l'usuari vegi d'una manera clara i entenedora quins moviments s'han detectat i en quin moment.

3.4.5 Smtplib.

El mòdul smtplib defineix un objecte de sessió de client SMTP (Simple Mail Transfer Protocol) que es pot utilitzar per enviar correus a qualsevol màquina d'Internet amb un programa resident d'escolta SMTP o ESMTP.

Una instància de SMTP encapsula una connexió SMTP. Té mètodes que admeten un repertori complet d'operacions SMTP i ESMTP. Si els paràmetres opcionals, host i port es proporcionen, es crida al mètode SMTP connect() amb aquests paràmetres durant la inicialització. Si aquesta crida a connect() retorna un codi que no es d'èxit, es genera un SMTPConnectError. [19]

La instància SMTP te diferents mètodes. [19] A continuació comentarem els utilitzats en el nostre treball.

- SMTP.ehlo(): tots els servidors requereixen utilitzar aquesta comanda per iniciar una salutació amb el altre servidor de correu, amb la finalitat de saber que tots els mètodes son suportats.
- SMTP.starttls(): col·loca la connexió SMTP en el mode TLS (Transport Layer Security), és a dir, seguretat de la capa de transport. Les posteriors comandes SMTP seran encriptades. Després d'executar aquesta comanda s'haurà de tornar a cridar a mètode ehlo().
- SMTP.login(user,password): és utilitzat per iniciar sessió en un servidor SMTP que requereix autenticació. El arguments són el nom d'usuari i la contrasenya per autenticar-se. Si la verificació a estat correcta, aquest mètode retornarà una normalitat, sinó, llançarà alguna de les excepcions establertes.
- SMTP.sendmail(from_addr,to_addrs,msg): mètode que serveix per enviar correus. Els arguments requerits són una cadena de direccions RFC 822 des d'on s'envia el missatge, una llista de direccions RFC 822 a les quals se'ls hi envia el correu i una cadena amb el missatge. El RFC 822 defineix un format

estàndard per a missatges electrònics, que consisteix amb un conjunt de camps d'encapçalament i un cos del missatge que és opcional. L'encapçalament conté informació sobre qui envia el missatge, així com per a qui es enviat, el tema .., entre d'altres. Si el cos del missatge està present, es separa del encapçalament mitjançant una línia en blanc. [20]
També cal dir que els paràmetres from_addr i to_addrs s'utilitzen per construir el sobre del missatge utilitzat pels agents de transport. En cap cas el SMTP modifica els encapçalaments del missatge.

- SMTP.close(): finalitza la sessió SMTP i tanca la connexió.

3.4.6 ConfigParser.

Aquest mòdul defineix la classe ConfigParser. Aquesta classe implementa un llenguatge bàsic d'analitzador d'arxius de configuració que proporciona una estructura similar a la que es trobaria als arxius INI de Microsoft Windows. És utilitzada per programes Python on l'usuari final pot personalitzar algun dels paràmetres.

El arxiu de configuració consta de seccions, que tenen el següent encapçalament: [secció] i seguides de entrades en format nom : valor (també nom = valor), amb continuacions en l'estil RFC 822. [21]

Aquest mòdul conté una classe anomenada RawConfigParser que serveix per crear l'objecte bàsic de configuració.

Les instàncies d'aquesta classe anteriorment anomenada tenen un seguit de mètodes. [21] Seguidament comentarem els utilitzats en el present treball.

- RawConfigParser.add_section(section): agrega una secció anomenada secció a la instància. Si ja existeix una secció amb aquest nom, es genera un error anomenat DuplicateSectionError.
- RawConfigParser.set(section,option,value): si la secció especificada existeix, es configura la opció donada amb el valor especificat. En cas contrari, es llança un error NoSectionError. Tot i que si que és possible utilitzar RawConfigParser per un emmagatzematge intern de valors que no siguin cadenes, la funcionalitat complerta, inclosa la interpolació i la sortida a arxius, només es pot aconseguir utilitzant una cadena de valors.
- RawConfigParser.write(fileobject): serveix per escriure una representació de la configuració en l'objecte d'arxiu especificat. Aquesta representació es podrà analitzar posteriorment mitjançant el mètode read().
- RawConfigParser.read(filename): intenta llegir i analitzar una llista de noms d'arxiu, retornant una llista amb els noms dels arxius que han estat analitzats amb èxit.
- RawConfigParser.get(section, option): agafa un valor de la opció per la secció anomenada.
- RawConfigPaser.getint(section, option): és un mètode de conveniència que força la opció de la secció especificada a ser un numero enter.

3.4.7 Os.path.

Aquest mòdul implementa algunes funcions útils en noms de ruta.

Un dels seus mètodes utilitzats és el os.path.isfile(path) que retorna cert si la ruta és un arxiu regular existent. [22]

3.4.8 Sys.

Aquest mòdul proporciona accés a algunes variables utilitzades o mantingudes per l'intèrpret i a funcions que interactuen fortament amb ell.

Un dels seus mètodes i utilitzat en el nostre treball és el sys.exit() i serveix per finalitzar el programa que s'està executant. S'implementa llençant l'excepció SystemExit, per la qual cosa es respecten les accions de neteja especificades per les clàusules finally de les declaracions try. [23]

4. Escenaris d'instal·lació.

4.1 Habitació dimensions reduïdes.

En quant als escenaris en els quals es podria instal·lar el nostre sistema n'hem plantejat dos de ben diferenciats.

Un d'ells és una habitació de dimensions reduïdes pensat per ser instal·lat en un àmbit més domèstic o d'unes característiques que requereixen el control de només una habitació per raons ben diverses.

Com ja vam dir anteriorment, el sensor pot ser col·locat en parets o sostres, segons convingui. Així doncs, hem decidit col·locar-los en la part superior de la paret, el més elevats possible i enfocant cap a la porta, principal font d'intrusos.

Sabent la posició elegida dels sensors, cal fer el càlculs per saber la mida màxima que l'habitació podria tenir. Tenint el compte en rang de detecció del sensor elegit i posant el cas que col·loquem un sensor a cada cantonada, és a dir, 4 sensors, podríem arribar a tenir unes mesures màximes de 9x10x2,7 m. Això ens permetria abastir tots els punts de l'habitació, sempre hi ha quan la lent de Fresnel ens permetés, donat el seu angle, no deixar cap punt mort.

Per al nostre treball, hem proposat una habitació de 20 m^2 (5 metres de llarg per 4 de ample), amb una altura de 2,7m. S'ha escollit aquesta mida per simular el que podria ser una habitació d'una casa habitual i poder així arribar a fer la implantació d'aquest sistema.

Els sensors PIR s'han d'evitar col·locar a prop de cossos que emetin rajos infrarojos que podrien provocar falsos positius en el nostre sensor. S'ha d'evitar doncs, col·locar-los en llocs on brilli directament la llum del sol, a prop de ventiladors que produeixin aire així com prop de calefactors. Com ja hem comentat, cal dir que és més beneficiós per a un sensor PIR si una persona, és a dir l'intrús, camina paral·lelament al sensor i no directament cap a ell.

Amb les mesures escollides i tenint en compte aquests factors, la disposició pensada dels diferents PIR és la següent:

- Suposant una habitació on la porta i una finestra estan oposades, la opció pensada és col·locar 2 sensors, un apuntant a la porta, de manera que el intrús al entrar hagi de travessar paral·lelament el sensor, i l'altre just damunt de la finestra, evitant així que hi brilli directament la llum del sol. Col·locant-los d'aquesta manera i degut a la mida de l'habitació abastaríem tots els punts d'aquesta, exceptuant alguns punts que, degut a les característiques dels sensors potser serien punt morts. En l'apartat de probes, s'aclariran aquests dubtes.

4.2 Espais més amplis.

Per procedir a la explicació del segon escenari pensat per aquest treball, cal que prèviament expliquem alguns conceptes.

El ESP8266 és un xip Wi-Fi de baix cost amb pila TCP / IP completa i capacitat de MCU (Micro Controller Unit). Aquest petit mòdul permet als microcontroladors connectar-se a una xarxa Wi-Fi i realitzar connexions TCP / IP senzilles.

Il·lustració 12. ESP8266

En les anterior imatges pot observar el ESP8266.

Cal comentar també el protocol de comunicació MQTT. Aquest és un protocol M2M (Machine to Machine) que serveix per a que les màquines es comuniquin entre elles. MQTT són sigles en anglès i signifiquen Message Queue Telemetry Transport. Està basat en un protocol de missatgeria publicació / subscripció, al contrari que HTTP que és petició / resposta.

Un dels seus punts forts és que és extremadament simple i lleuger. Per aquest motiu és molt interessant per a sistemes que requereixen poc ample de banda, tenen una alta latència (temps des que s'envia un missatge fins que arriba al seu destí) i requereixen de poc consum dels dispositius. [24]

El que pretenem fer és el disseny d'un sistema com el anterior però que es pugés implantar en espais més grans, com podria arribar a ser la totalitat d'una casa. Així doncs, s'ha pensat en la següent estratègia de resolució: es tindran els diferents PIRs connectats cadascun a un ESP. Des d'aquests es dispositius s'enviaran les senyals d'alerta via Wi-Fi utilitzant el protocol MQTT.

Aquesta és una solució bastant òptima ja que s'evita haver de cablejar tota la instal·lació amb cables de longitud elevada i aconseguim el resultat desitjat.

Tot i els avantatges que aquesta solució presenta, finalment no hem acabat implementant-la i ens hem centrat en el primer escenari anteriorment mencionat.

5. Consum energètic del sistema.

Com ja hem comentat en la introducció, un dels factors a tenir en compte de manera bastant significativa és el consum energètic, ja que en sistemes que es suposa que han de funcionar 24 hores al dia, es important que sigui el més baix possible.

Així doncs, és per aquest motiu que tant el microordinador com els sensors escollits són de baix cost, per intentar que aquest consum sigui el més reduït possible.

Sabem que la Raspberry Pi, s'alimenta a 5'1 volts i que la intensitat màxima que necessita es 2'5 Amperes, per assegurar que podem connectar els 4 ports USB, així com altres perifèrics, a més del propi consum. Anem a suposar que aquesta es la màxima intensitat, doncs, que pot, consumir, cosa que a la pràctica mai succeirà perquè estaríem portant a microordinador al màxim de les seves capacitats.

Només a efectes de càlcul:

$$P = V \cdot I = 5'1 \cdot 2'5 = 12'75W$$

Si volem saber quant gastaria en un any:

$$12'75\ W \cdot 24\frac{h}{dia} \cdot 30\frac{dia}{mes} \cdot \frac{12mesos}{1\ any} = 110'16\frac{kWh}{any}$$

Informant-nos del preu del kilowatt hora, podem saber el preu d'alimentar una Raspberry Pi en un any. Segons fonts utilitzades en aquest treball [25], el preu fixe del kWh està actualment, des del 15 de Juliol de 2018, en: 0'146632 €/kWh.

Per tant, el preu a pagar seria:

$$TOTAL = 110'16\frac{kWh}{any} \cdot 0'146632\frac{€}{kWh} = 16'15\frac{€}{any}$$

Aquest preu es suposant, com ja hem dit, que treballem utilitzant el màxima amperatge que ens pot subministrar la Raspberry Pi, però, com es obvi, no és el que utilitzem nosaltres en aquest projecte. Només tenim dos sensors de presencia connectats, que, a més, són de baix consum així com també de baix cost.

El sensor escollit té un consum mitjà d'1mA i sabent que n'utilitzem dos, podem dir que, afegint el consum de la nostra Raspberry Pi, que de mitjana, en una situació d'estrès, pot consumir 0'85A, tindríem:

$$P = V \cdot I = 5'1 \cdot 0'852 = 4'35W$$

Si volem saber quant gastaria en un any:

$$4'35\ W \cdot 24\frac{h}{dia} \cdot 30\frac{dia}{mes} \cdot \frac{12mesos}{1\ any} = 37'54\frac{kWh}{any}$$

Per tant, el preu a pagar seria:

$$TOTAL = 37'54\frac{kWh}{any} \cdot 0'146632\frac{€}{kWh} = 5'50\frac{€}{any}$$

Podem observar que el consum i per tant, el cost econòmic es reduït i per tant, assolim així l'objectiu d'obtenir un sistema de vigilància de baix cost.

6. Plantejament de la solució de monitoratge.

6.1 Familiarització amb el sensor HC-SR501

S'han realitzat unes proves inicials amb la intenció de conèixer de primera mà el funcionament del sensor estudiat fins ara de manera teòrica. Amb el codi que es mostra a continuació, s'han anat variant els potenciòmetres esmentats anteriorment amb la finalitat d'entendre millor el funcionament d'aquest.

Com s'observa el llenguatge de programació escollit es Python.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD) #Set GPIO to pin numbering
pir = 13 #Assign pin 8 to PIR
GPIO.setup(pir, GPIO.IN) #Setup GPIO pin PIR as input
print ("Sensor initializing . . .")
time.sleep(2) #Give sensor time to startup
print ("Active")
print ("Press Ctrl+c to end program")

try:
    while True:
        if GPIO.input(pir) == True: #If PIR pin goes high, motion is detected
            print ("Motion Detected!")
            time.sleep(0.1)
except KeyboardInterrupt: #Ctrl+c
    pass #Do nothing, continue to finally

finally:
    GPIO.cleanup() #reset all GPIO
    print ("Program ended")
```

Il·lustració 13. Programa inicial de proba.

Així doncs s'observa que, durant el temps que l'alarma està activa es genera contínuament un avís de que s'ha detectat moviment. En el cas de que el mode seleccionat amb el jumper sigui el mono-tret, com ja s'havia comentant, si ocorre un altre esdeveniment durant el temps d'activació de l'alarma, aquest no serà considerat. Per fer proves, s'ha col·locat el temps d'activació al mínim, 3 segons, per així poder observar que cada moviment era detectat correctament.

Amb el cas de trets repetitius, si durant els temps d'activació de l'alarma es detecta un altre esdeveniment, es sumarà el temps d'activació al que ja havia transcorregut, i aquest serà el nou temps d'activació de l'alarma.

S'han fet algunes proves més. Suposant que col·loquem el temps d'activació d'alarma en aproximadament un minut. En el mode mono-tret, quan detecta moviment, manté la sortida activa durant un minut, independentment del que passi, és a dir, de si hi ha més moviments o no.

En canvi, en el mode de trets repetitius, cada esdeveniment que es vagi detectant mentre l'alarma esta activa, serà detectat. S'evidenciarà que s'ha detectat afegint el temps des de l'inici fins que s'ha detectat aquest moviment al temps d'alarma activa triat. Així doncs, si passats 20 segons es detecta un altre esdeveniment, se li suma el minut d'alarma activa escollit i llavors són 80 segons d'alarma activa.

Com vam comentar en l'explicació d'aquests sensor elegit, té dos modes de treball. Un que seria el mono-tret i el altre, tret repetitiu. En el mode que només hi ha 1 sol tret, si durant el temps d'activació de l'alarma es detectat un altre esdeveniment, aquest no serà considerat.

En el nostre cas, hem configurat el temps d'activació en el mínim, és a dir, 3 segons, i en el mode mono-tret. És a dir, cada 3 segons com a mínim podrem detectar un únic moviment i quan així sigui, sens informarà.

Tenim clar que el temps d'activació de l'alarma ha de ser el mínim possible, perquè sinó, en ambdós modes, si per exemple configurem aquest temps d'activació en 1 minut, durant tot el minut estarem rebent sortida activa que per tant, farà que el programa dissenyat que a continuació explicarem, informi tota l'estona de que hi ha moviment.

Hem escollit el mode d'un únic tret ja que si escollim l'altre mode, el tret repetitiu, cada esdeveniment que succeís, quan no hages transcorregut el temps d'activació de l'alarma, seria detectat i el que faria seria augmentar aquest temps i, com ja hem dit, interessa que aquet sigui el més baix possible. Llavors, a l'hora de notificar-ho no seria real ja que realment només es tractaria d'un moviment i estaríem enviant alertes continues com si hagessin succeït molts esdeveniments.

Cal dir, també que el mode d'un únic tret, pot ser útil en cas de que es desitgi encendre un llum, per exemple, en haver detectat presencia humana. En aquest cas no caldria que el sensor detectes cada moviment, sinó que es quedes activat durant un període, el del temps seleccionat d'alarma activa.

En canvi, en el cas de voler detectar intrusos i fer sonar un brunzidor si que és necessari que quan s'ha detectat un moviment i durant el temps d'activació de l'alarma, independentment del que succeeixi, és a dir, de si hi ha més moviments o no, tota l'estona es doni la sortida activa per a que el brunzidor soni sense parar. En aquest cas, cada moviment que es vagi detectant, com s'ha explicat, s'anirà acumulant fen així augmentar el temps de sortida activa i mantenint el brunzidor activat. En aquest cas interessaria col·locar el temps d'alarma el més elevat possible.

En aquest projecte, tot i haver donat als sensors PIR la funció de detectar intrusos, hem escollit el mode mono-tret perquè en el nostre cas no volem fer sonar un brunzidor sinó que volem alertar a l'usuari del sistema. Per tant, no necessitem que s'acumulin els moviments i anar allargant així el temps d'alarma activa.

El temps d'activació de l'alarma escollit, 3 segons, i el disseny del programa que seguidament explicarem, és el necessari per avisar a l'usuari un sol cop, juntament així amb el mode mono-tret seleccionat.

6.2 Solució de control detallada.

Com es va comentar al principi del document, un dels requeriments era dissenyar la nostra aplicació de control per utilitzar-la amb el microordinador Raspberry Pi. Com ja hem vist en les probes inicials, hem decidit escollir el llenguatge de programació Python.

Dins del llenguatge de programació, les classes proveeixen una forma d'empaquetar dades i funcionalitat junts. En crear una nova classe, es crea un nou tipus d'objecte, permetent crear noves instàncies d'aquest tipus. Cada instància de classe pot tenir atributs adjunts per mantenir el seu estat. Les instàncies de classe també poden tenir mètodes ,definits per la seva classe, per modificar el seu estat. [26]

Així doncs, hem estructurat el codi en diferents classes, ajudant-nos també de l'herència que aquestes poden tenir. L'execució d'una classe heretada procedeix de la mateixa manera que una classe base. Quan l'objecte classe es construeix, es té en compte a la classe base. Això es fa servir per resoldre referències a atributs: si un atribut sol·licitat no es troba a la classe, la recerca continua per la classe base. Aquesta regla s'aplica de manera recursiva si la classe base deriva d'alguna altra classe.

Hi ha 3 carpetes que contenen les 3 funcionalitats del nostre sistema. La primera de elles és la carpeta que conté els diferents mòduls per poder enviar notificacions. En ella hi ha l'arxiu *main*, en el qual es fa és indicar quin servidor volem utilitzar (en el nostre cas Gmail) així com també es segueix el protocol per poder-nos connectar. En l'altre fitxer que hi ha, fem el login per guanyar accés i preparem la capçalera en una classe que agrupa totes aquestes funcionalitats.

La segona carpeta conté el mòdul anomenat *pir.py* que el fa es analitzar, en una classe, quan pins hi ha a la llista de pins a utilitzar establerts per l'usuari i els configurem com una entrada. A més, es detecta si hi ha moviment o no. Finalment, en la darrera carpeta, hi ha contingut les diferents *templates* (plantilles) emprades per dissenyar el servidor web que l'usuari podrà consultar.

Hi ha dos fitxers més que també cal comentar. Un d'ells és el fitxer de configuració on l'usuari configura diversos paràmetres que seguidament explicarem i l'altre és el fitxer de text on es guarden els contactes als quals se'ls hi vol enviar l'alerta via Gmail.

Com és obvi, hi ha dos mòduls principals a part de tot el anteriorment explicat. Un d'ells és el programa principal, estructurat en un classe, que agrupa en diferents funcions, el que anteriorment s'ha anat creant en mòduls diferents. L'altre és l'aplicació feta amb Flask que ens permet llençar el servidor web.

Dels dos escenaris dissenyats, finalment, per manca de temps i degut al volum de feina que ha suposat fer la implementació de tot el codi per poder monitorar el sistema, només se n'ha implementat un, el primer d'ells, que consistia en connectar diversos PIRs a la Raspberry. Per tant, el nombre de PIRs utilitzats serà dos. Des del programa central de monitoratge es cridaran totes les funcions que permeten les diferents funcionalitats del nostre programa.

Resumint breument les funcionalitats del programa creat podríem dir que: un cop el sensors PIR escollits detecten canvi en la radiació rebuda, el que es fa és emmagatzemar la detecció en un fitxer de text, així com en una base dades.

Guardant-ho en aquesta darrera, permetem que l'usuari ho pugi consultar des d'un servidor web. Finalment s'envia una senyal d'alerta via Gmail.

A continuació, la solució de control s'explicarà amb més detall analitzant cada part degudament.

Primerament cal dir que s'ha creat un fitxer de configuració mitjançant el mòdul ConfigParser. En aquest fitxer, l'usuari final pot triar els GPIOs de connexió dels dos sensors PIR utilitzats. Cal especificar-li, que el mode utilitzat és el BOARD, per a que així pugi fer així una correcta configuració. En aquest arxiu, a més, l'usuari també a d'introduir l'usuari i la contrasenya que s'utilitzaran per iniciar sessió al correu des del qual s'enviaran les notificacions en haver-se detectat moviment. Finalment, haurà d'introduir el mode per saber segons quin criteri vol rebre les diferents notificacions produïdes. Els dos modes implementats són els següents:

1. El primer mode envia un missatge d'alerta a la conta del servidor SMTP de Gmail cada vegada que es detecta un moviment.

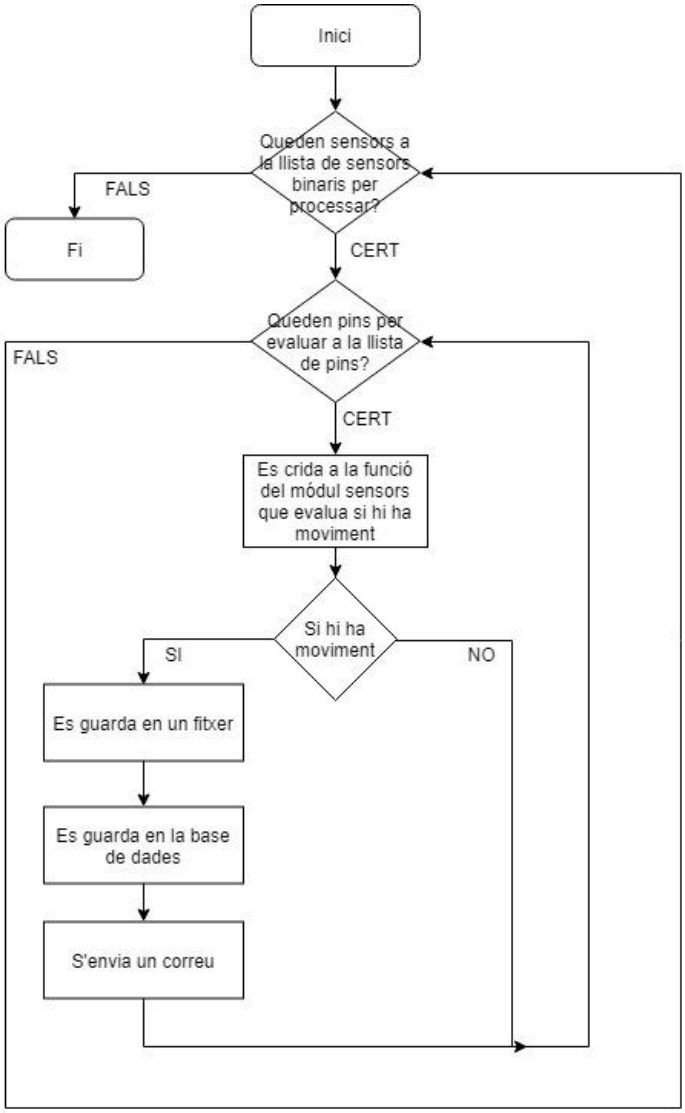
2. En canvi, el segon mode és una mica més elaborat. En cas de que entre dos moviments hi hagi menys de 20 segons, no s'envia l'alerta, és van agrupant en una llista a fi d'enviar un mateix correu considerant moviments molt seguits, un mateix esdeveniment. Així doncs, tots el moviments consecutius que hi hagués amb menys de 20 segons entre ells, serien considerants un mateix esdeveniment. El que fem llavors és esperar a que transcorri un dels fets que poden succeir a continuació:

    - Un d'ells és que es produeixi un altre moviment en menys de 20 segons, però que el temps total des del primer moviment fins al darrer sigui major a 40 segons. En aquest cas es considerarà que passats 40 segons tot i ser un mateix esdeveniment, ja és necessari informar a l'usuari i per tant s'enviarà l'alerta informant de quants moviments en quant temps han tingut lloc i indicant també quanta estona fa d'aquests moviments.

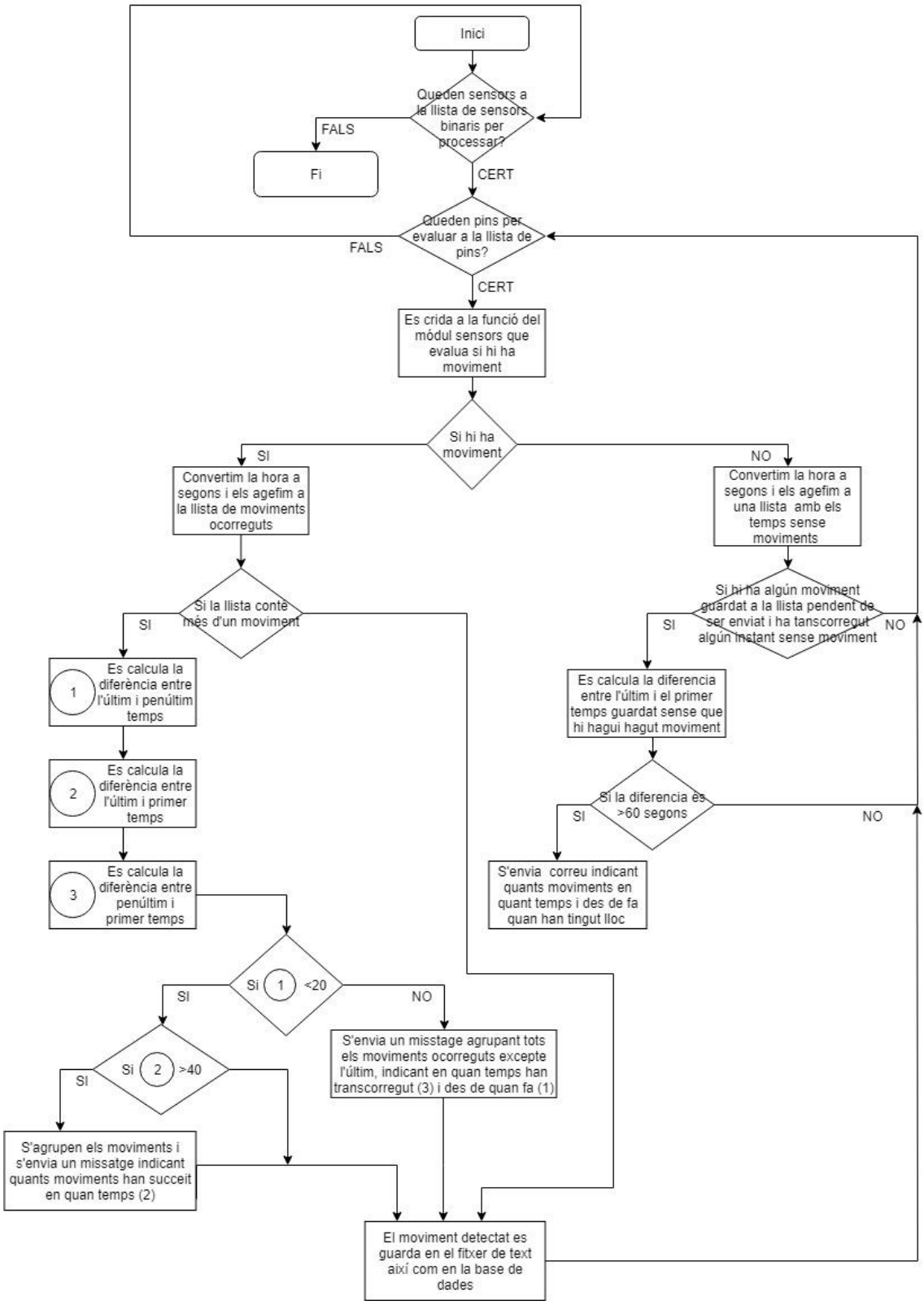
    - Un altre és que es produeixi un altre moviment amb més de 20 segons de diferencia, cosa que provocarà l'enviament del missatge que contindrà tots els moviments anteriors a aquest últim. Llavors, doncs, s'enviarà un missatge indicant quants moviments s'han detectat i fent saber durant quants segons s'han produït aquests moviments. A més, també s'indicarà quan temps fa d'aquests grup de moviment. L'últim moviment romandrà a l'espera d'un següent moviment.

- Finalment l'altra cosa que pot passar es que transcorri un minut sense tenir cap altre moviment. El que farem en aquest cas serà enviar el missatge ja que considerem que a transcorregut massa temps sense haver detectat moviment i l'alerta no pot romandre a l'espera d'un següent moviment.

L'algorisme del que s'acaba d'explicar, és a dir, de les diferents maneres en que l'usuari pot rebre les notificacions, queda de manifest d'una manera més tècnica en els següents diagrames de flux. Es mostren, el primer i el segon mode de ser notificat, respectivament.



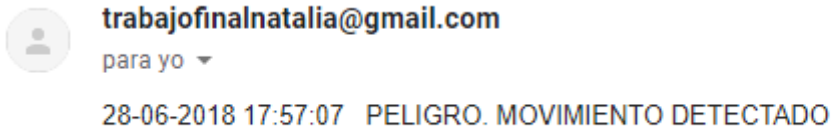
Il·lustració 14. Diagrama de flux del primer mode de notificacions.



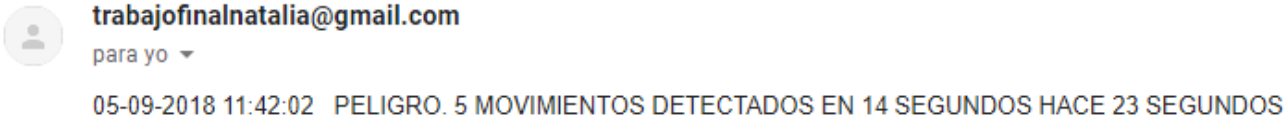
Il·lustració 15. Diagrama de flux del segon mode de notificacions.

Si bé és cert que en els diagrames de flux anteriors s'observa que quan han completat el cicle i ja no queden ni més pins ni més sensors a analitzar la seqüencia acaba, en el programa principal el que hem fet ha estat introduir aquestes funcions en un bucle infinit per a que es repeteixi sempre sense fi.

La manera en que l'usuari veu els avisos al seu correu segons el primer o el segon mode és la que es veu en les següents imatges respectivament.



Il·lustració 16. Visió correu electrònic mode 1.



Il·lustració 17. Visió correu electrònic mode 2.

El fitxer de configuració esmentat té el següent aspecte:

configuration.cfg

```
[GPIO BOARD]
Pin 1 :
Pin 2 :

[Notifications]
User email :
Password :

[Notifications mode]
# Mode 1: if you want to be notified every time.
# Mode 2: if you want to be alerted with notifications ghatered.
Mode :
```

Il·lustració 18. Aspecte del fitxer de configuració.

Les probes que hem realitzat en aquest sistema de seguretat, s'han realitzat amb dos sensors PIR, però s'ha estructurat el codi utilitzat, en general, i més concretament la part del fitxer de configuració, de manera que en cas de que l'usuari volgués introduir un altre

sensor PIR només hauria d'afegir una opció més, indiciant el pin en mode board que vol utilitzar. Això fa ampliable el nostre sistema de seguretat i no restringim així el numero de PIRs a utilitzar.

En cas de que hi hagi una de les seccions del fitxer de configuració que no s'hagi omplert, el programa central guardian.py avisarà i llençarà una excepció SystemExit, finalitzant així el programa. Si aquest fitxer no existeix, es crearà i s'avisarà a l'usuari de que l'ha d'omplir degudament. Un cop no falti cap opció a emplenar en el fitxer de configuració, l'usuari podrà tornar a executar el programa principal sense cap problema.

Per enviar correus s'ha utilitzat el servidor SMTP ja que és el estàndard de Internet més utilitzat a dia d'avui. Tothom pot disposar, ja sigui en l'ordenador, tableta o mòbil personal d'un servidor d'aquestes característiques que li permeti l'accés al correu electrònic.

Cal saber que la conta des de la que s'enviïn el missatges ha de permetre que aplicacions menys segures utilitzin aquesta conta. Aquest és un factor que l'usuari haurà de tenir en compte per a un correcte funcionament. Permetre això en una conta de Gmail es molt senzill. Només s'ha d'anar a la conta en la qual es vol permetre que aplicacions menys segures l'utilitzin i a l'apartat d'Inici de sessió i seguretat es trobarà un apartat anomenat Aplicacions amb accés a la conta. Al final d'aquell apartat trobarem el botó per permetre aquest accés.

Hi ha un altre fitxer en el qual es guarden els contactes, separats per un espai en blanc i tots en la mateix línia, als qual se'ls hi desitja enviar el correu d'avís. En cas de que el arxiu no existeixi es crea i s'avisa a l'usuari de com a d'introduir-hi les diferents adreces.

Un cop l'usuari final ja ha configurat tots els paràmetres variables depenent de cada usuari, ja es pot començar a utilitzar l'aplicació.

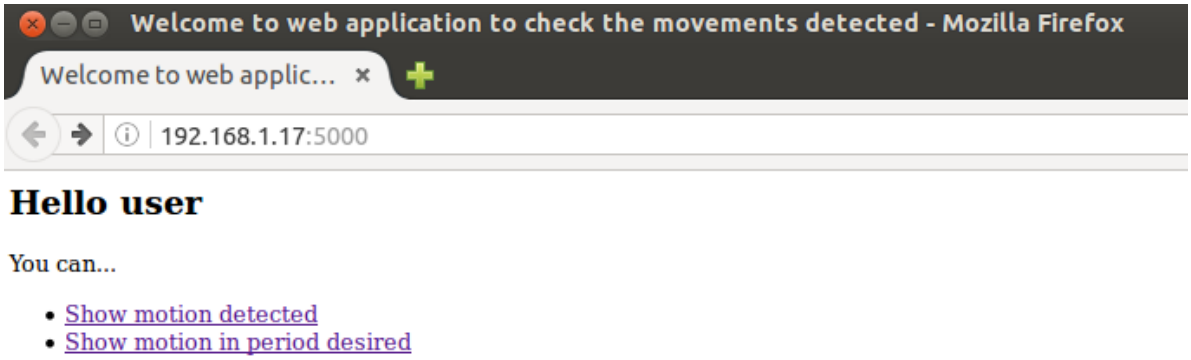
Quan es detecta moviment en l'habitació on s'ha instal·lat el sistema es produeix un canvi en l'estat dels sensors, i el que fem es enviar, com ja hem dit, una alerta des d'una conta de correu de Gmail. A més, les dades enregistrades son emmagatzemades en un fitxer de text així com també en una base de dades. Aquesta base de dades, haurà d'estar creada prèviament i haurà de tenir també creada la taula amb les diferents columnes que utilitzarem prèviament definides.

La base de dades que hem elegit es SQLite3. Aquesta funciona correctament en llocs de tràfic mitjà-baix. Si bé es cert que suporta un número il·limitat de lectors simultanis, només està permès que un únic usuari modifiqui dades. Així doncs, tot i que no es una base de dades de gran envergadura, es idònia per utilitzar-la com a base de dades interna o temporal. A més, és simple d'instal·lar i utilitzar i per tant és ideal per començar a familiaritzar-se amb una base de dades. Es per aquests motius que l'hem elegit.

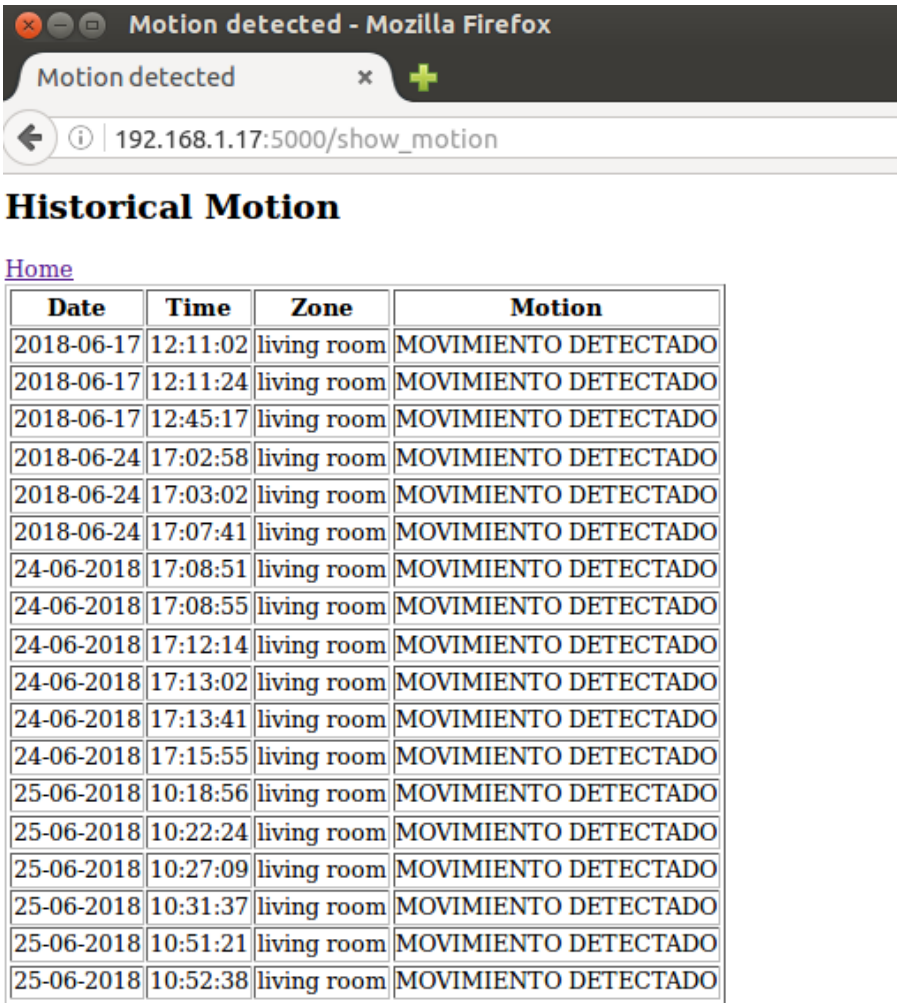
Si l'usuari desitja veure un resum de tots els moviments detectats, executant el fitxer moviment.py que el que fa és llençar un servidor web fet amb Flask, ho podrà fer. Haurà de clicar el link que apareix al principi, que és on l'aplicació estarà corrent i allí podrà veure-ho de dues maneres. Una d'elles serà veient tots els moviments enregistrats des de que aquell sistema de seguretat s'ha posat en funcionament. L'altra opció serà elegint

un període de dates entre les quals vol observar les diferents alertes i això serà el que se li mostrarà. Aquesta segona opció permet a l'usuari més llibertat a l'hora de poder elegir quin període de dades l'interessa veure i no haver de buscar per tota la llista, cosa que fa més interesant l'aplicació web.

En les següents imatges es mostra la interfície que l'usuari veu quan es disposa a observar els moviments detectats. Es veurà també com hem implementat la opció de poder tornar a la pàgina inicial, la que es pot veure seguidament, quan estiguem visualitzant les dades.



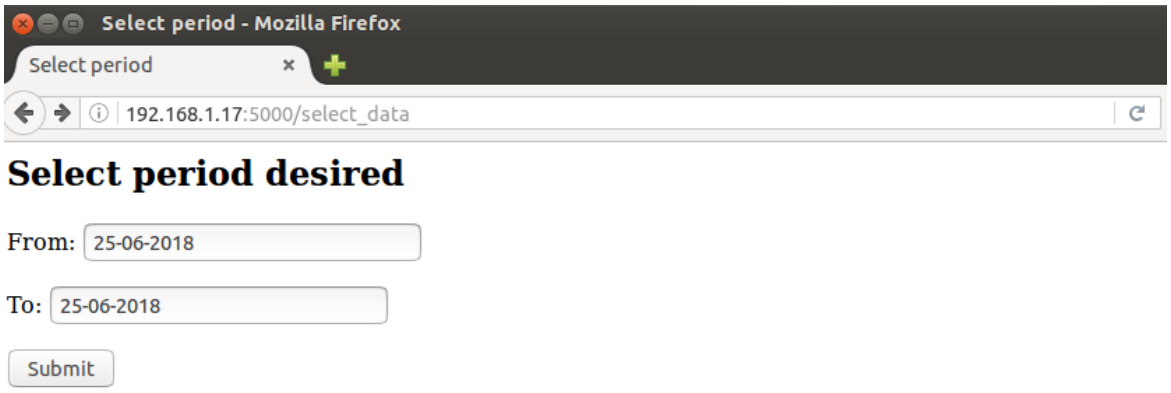
Il·lustració 19. Índex servidor web on es mostren les diferents funcionalitats.



Date	Time	Zone	Motion
2018-06-17	12:11:02	living room	MOVIMIENTO DETECTADO
2018-06-17	12:11:24	living room	MOVIMIENTO DETECTADO
2018-06-17	12:45:17	living room	MOVIMIENTO DETECTADO
2018-06-24	17:02:58	living room	MOVIMIENTO DETECTADO
2018-06-24	17:03:02	living room	MOVIMIENTO DETECTADO
2018-06-24	17:07:41	living room	MOVIMIENTO DETECTADO
24-06-2018	17:08:51	living room	MOVIMIENTO DETECTADO
24-06-2018	17:08:55	living room	MOVIMIENTO DETECTADO
24-06-2018	17:12:14	living room	MOVIMIENTO DETECTADO
24-06-2018	17:13:02	living room	MOVIMIENTO DETECTADO
24-06-2018	17:13:41	living room	MOVIMIENTO DETECTADO
24-06-2018	17:15:55	living room	MOVIMIENTO DETECTADO
25-06-2018	10:18:56	living room	MOVIMIENTO DETECTADO
25-06-2018	10:22:24	living room	MOVIMIENTO DETECTADO
25-06-2018	10:27:09	living room	MOVIMIENTO DETECTADO
25-06-2018	10:31:37	living room	MOVIMIENTO DETECTADO
25-06-2018	10:51:21	living room	MOVIMIENTO DETECTADO
25-06-2018	10:52:38	living room	MOVIMIENTO DETECTADO

Il·lustració 20. Moviments detectats en la data i hora transcorreguts.

Observant detalladament veiem que les primeres dates tenen un format diferent a les darreres degut a que en comptes de la funció localtime() en un principi utilitzàvem la date.today() importada del mòdul datetime.



Il·lustració 21. Selecció de períodes en servidor web.

Date	Time	Zone	Motion
25-06-2018	10:18:56	living room	MOVIMIENTO DETECTADO
25-06-2018	10:22:24	living room	MOVIMIENTO DETECTADO
25-06-2018	10:27:09	living room	MOVIMIENTO DETECTADO
25-06-2018	10:31:37	living room	MOVIMIENTO DETECTADO
25-06-2018	10:51:21	living room	MOVIMIENTO DETECTADO
25-06-2018	10:52:38	living room	MOVIMIENTO DETECTADO

Il·lustració 22. Resultat cerca filtrada realitzada en la imatge anterior.

El codi del nostre projecte consta de dos mòduls secundaris ben diferenciats: un de notificacions i l'altre de sensors. Aquests dos mòduls són posteriorment importats al mòdul principal, anomenat en el nostre projecte guardian.py. Ambdós mòduls secundaris tenen un fitxer anomenat __init__.py que es la manera que te Python d'inicialitzar paquets, mòduls o llibreries. Li diu al sistema quines carpetes contenen aquests elements per poder importar-los directament des de qualsevol nivell del nostre projecte.

El codi s'ha estructurat d'e manera que ens permeti llibertat a l'hora de fer modificacions en ell. Com ja s'ha dit, si el volgués ampliar el nombre de sensors utilitzats, per exemple, l'usuari només hauria d'introduir-lo en el fitxer de configuració i no s'hauria de modificar cap línia del codi.

A més, amb els diferents mòduls constituïts, obtenim un ordre que per a posterior modificació, millora o simplement lectura del codi ens facilita molt les coses.

7. Probes del sistema implementat.

En l'apartat dels escenaris d'instal·lació ja vam comentar la disposició que tindria el nostre sistema de vigilància. Així doncs el que hem fet ha estat unes petits probes per veure la eficiència d'aquest sistema dissenyat. En el annexes es podrà veure una imatge del escenari emprat per a les probes així com la disposició dels sensors PIR.

Les probes han estat molt senzilles només amb la finalitat de detectar que el sistema funciona correctament i que no deixa o deixa els mínims, punts morts. Les diferents probes realitzades han estat les següents:

- Passar per fora de l'habitació, molt a prop de la porta per veure si ens detecta. De manera encertada no hem rebut resposta per part del sistema.
- Entrar a l'habitació una distància molt petita d'uns 30 cm per veure si hem sigut detectats, i així ha estat.
- Fer entrar a un gos, així com també entrar nosaltres gatejant. Ambdós hem sigut correctament detectats.
- Amb més d'una persona entrat també em estat detectats correctament, i al moure'ns per dins els moviments dels dos han sigut detectats.

Un cop dins de l'habitació hem fet probes per veure si algun de les zones accessibles quedava fora del rang de detecció. Un dels problemes que teníem al principi era que si ens col·locàvem al centre de l'habitació ambdós sensors ens detectaven, ja que tenim la distància de detecció al màxim. Així doncs, vam optar per col·locar cada potenciòmetre que et permet regular la distància de detecció a aproximadament, a 2'5m per evitar que els dos sensors ens detectessin a la vegada.

En l'escenari elegit, degut al mobiliari que el que fa es reduir la zona on l'intrús pot accedir, no hi havia cap punt on no fossis detectat, però de ben segur que en una habitació buida si que quedarien punts morts.

Així doncs, després d'aquestes petits probes realitzades podem concloure dient que el sistema dissenyat i posteriorment implementat és de qualitat.

8. Conclusions.

Responent a l'objectiu principal d'aquest projecte, que demanava el disseny i la posterior implementació d'un sistema de vigilància mitjançant sensors de presència, capaç de detectar qualsevol moviment en una zona delimitada i generar els corresponents avisos, cal dir que, com s'ha pogut observar, ha quedat complert.

Així doncs, s'ha aconseguit d'una manera satisfactòria poder connectar a la Raspberry Pi, microordinador emprat ja que era un dels requeriments, els diferents sensors passius de presència per poder detectar els diferents moviments no previstos en una habitació en concret i alertar-ne d'això.

Cal dir que la creació d'aquesta treball m'ha permès profunditzar en programació, sobretot en el llenguatge Python, emprat en aquest projecte, així com també en HTML, utilitzat a l'hora de crear el servidor web que permet a l'usuari la consulta dels diferents esdeveniments ocorreguts. A més, també m'ha permès desenvolupar-me en l'àmbit de comunicacions permetent així poder alertar de forma remota a l'usuari dels diferents moviments detectats.

8.1 Futures millores i ampliacions.

Un dels objectius que s'havien marcat com a secundaris, va estar la possible alimentació d'aquest sistema d'una manera alternativa. Es un aspecte que, posteriorment, degut a la manca de temps, no s'ha pogut estudiar.

Aquest fet és molt interessant ja que en ser un sistema de baix cost que s'ha de mantenir actiu les 24 hores del dia, seria una bona idea que estigues alimentat, per exemple, amb plaques solars. Aquesta podria ser una millora força interessant a realitzar en aquest projecte.

Com ja s'ha comentat al llarg del treball, el servidor web implementat només és possible executar-lo en un servidor local i per tant no seria permès l'accés des de l'exterior. Una millora que es podria fer en aquest projecte seria que es poguessin fer consultes des d'un servidor extern. Aquesta millora hauria d'incloure algunes mesures de seguretat, perquè sinó qualsevol podria consultar les nostres dades privades. Així doncs, s'hauria d'introduir un usuari i contrasenya que s'haurien de validar per poder accedir a l'aplicació web.

Una ampliació a aquest treball seria la implantació d'una càmera que permetés enregistrar d'una forma visual els diferents esdeveniments que es van produint. Això permetria a més que l'usuari pugues detectar falsos positius o tenir una prova física de la possible intrusió.

9. Referències.

[1] https://pinout.xyz/pinout/uart

[2] https://es.wikipedia.org/wiki/Universal_Asynchronous_Receiver-Transmitter

[3] https://franciscomoya.gitbooks.io/taller-de-raspberry-pi/content/es/elems/i2c.html

[4] Derek Molloy. Real-Time Interfacing Using the Arduino. En: Wiley. Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux. Edició 1. 2016. 453-480.

[5] https://es.wikipedia.org/wiki/Serial_Peripheral_Interface

[6] https://pinout.xyz/pinout/spi

[7] http://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html

[8] https://www.lifewire.com/selecting-between-i2c-and-spi-819003

[9] https://www.kernel.org/doc/Documentation/w1/w1.generic

[10] https://puntoflotante.net/MANUAL-DEL-USUARIO-SENSOR-DE-MOVIMIENTO-PIR-HC-SR501.pdf

[11] https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/overview

[12] http://www.holtek.com/documents/10179/116711/HT7M21x6v130.pdf

[13] https://wiki.up-community.org/RPi.GPIO

[14] http://www.mnp.cl/post/que-es-board-bcm-raspberry-pi

[15] https://docs.python.org/2/library/time.html

[16] https://www.sqlite.org/about.html

[17] https://www.essentialsql.com/sqlite3-review-great-for-beginners-and-those-learning-sql/

[18] http://f-guitart.github.io/progcoms3-flask

[19] https://docs.python.org/2/library/smtplib.html

[20] https://msdn.microsoft.com/en-us/library/ms527253(v=exchg.10).aspx

[21] https://docs.python.org/2/library/configparser.html

[22] https://docs.python.org/2/library/os.path.html

[23] https://docs.python.org/2/library/sys.html

[24] https://programarfacil.com/esp8266/mqtt-esp8266-raspberry-pi/

[25] https://www.endesaclientes.com/articulos/tarifas-reguladas-luz-gas.html

[26] http://docs.python.org.ar/tutorial/2/classes.html

10. Annexes.

10.1 Codi creat i utilitzat.

GMAIL

__init__.py

```
1. from .main import Notification
2. from .sendemail import SendMail
3.
4. __all__ = ["SendMail","Notification"]
```

main.py

```
1. import smtplib
2.
3. class Notification(object):
4.     def __init__(self, username, password):
5.         self.username = username  # Username
6.         self.password = password  # Password
7.         SMTP_SERVER ='smtp.gmail.com'#SMTP_SERVER
8.         SMTP_PORT = 587           # SMTP_PORT
9.         self.smtpserver = smtplib.SMTP(SMTP_SERVE
  R, SMTP_PORT)  # Creation of SMTP server object.
10.
11.         # Say hello (ehlo) to smtp server in
  order to follow the protocol.
12.         self.smtpserver.ehlo()
13.         self.smtpserver.starttls()
14.         self.smtpserver.ehlo
```

sendemail.py

```
1. import smtplib                       # smtp - Simple
   Mail Transport Protocol - library for sending
   email.
2. from .main import Notification
3.
4. class SendMail(Notification):    # This class is
   created for send in an automatically mesage using
   Simple Mail Transfer Protocol (SMTP).
5.     def __init__(self, username, password):
6.         Notification.__init__(self, username, pas
   sword)
7.
8.     def login(self):
9.         self.smtpserver.login(self.username,self.
   password)
10.
11.     def sendemail(self, recipient, subject, msgbo
   dy):
12.         #Prepare email header information and
   message body.
13.         header = 'To:' + recipient + '\n' + 'From:
   ' + self.username + '\n'
14.         header = header + 'Subject: ' + subject
   + '\n'
15.         msg = header + '\n' + msgbody + ' \n\n'
16.         self.smtpserver.sendmail(self.username, re
   cipient, msg) # Send email to recipicient user.
17.
18.     def closeSMTP(self):
19.         self.smtpserver.close() # Close smtp
   server.
```

SENSORS

__init__.py

```
1. from .main import Sensor
2. from .binary import BinarySensor
3. from .pir import PIR
4.
5. __all__ = ["Sensor", "BinarySensor", "PIR"]
```

main.py

```
1. class Sensor(object):
2.     """Main Sensor class"""
3.     def __init__(self, name):
4.         super(Sensor, self).__init__()
5.         self.name = name
```

binary.py

```
1. from .main import Sensor
2.
3. class BinarySensor(Sensor):
4.     """BinarySensor"""
5.     def __init__(self, name):
6.         super(BinarySensor, self).__init__(name)
7.         self.state = False
8.
9.     def get_state(self):
10.        return self.state
11.
12.    def update_state(self):
13.        pass
```

pir.py

```
1. from .binary import BinarySensor
2. import time
3. from time import strftime, localtime
4. import RPi.GPIO as GPIO
5.
6. class PIR(BinarySensor):
7.     def __init__(self, name, pin_list):
8.         super(PIR, self).__init__(name)
9.         GPIO.setmode(GPIO.BOARD)
10.    # For each pin stored in pin_list set it up
   as an input.
11.    for pin in pin_list:
12.        self.pin = pin
13.        GPIO.setup(pin, GPIO.IN, pull_up_down = GP
   IO.PUD_DOWN) # Used to avoid high output when
   nothing is connected.
14.
15.    # If GPIO state is HIGH, return the time when
   moviment was detected.
16.    def update_state(self,pin):
17.        if GPIO.input(pin):
18.            timex = strftime("%H:%M:%S", localtime(
   ))
19.            value = 1
20.            time.sleep(1) # For not compare too
   fast.
21.        else:
22.             timex = 0
23.             value = 0
24.        return timex, value
```

TEMPLATES

functionalities.html

```
<title>Welcome to web application to check the movements detected</title>

  <h2>Hello user</h2>
  <p>You can...</p>
  <ul>
    <li><a href="/show_motion">Show motion detected</a></li>
    <li><a href="/select_data">Show motion in period desired</a></li>
  </ul>
```

select_data.html

```
<head>
  <title>Select period</title>
</head>
    <body>
      <h2>Select period desired</h2>
      <form action="/select_data" method="post">
        <p>From: <input type="date" name="date1"></p>
        <p>To: <input type="date" name="date2"></p>
    <input type="submit" value="Submit">
     </form>
    </body>
```

show_motion_desired.html

```
<h2>Historical Motion</h2>

<head>
    <title>Motion desired</title>
</head>

<table border="1">
    <tbody><tr>
<th>Date</th>
<th>Time</th>
<th>Zone</th>
<th>Motion</th>

</tr>
{% for row in motion_desired %}
<tr>
{% for cell in row %}

  <td> {{cell}} </td>

{% endfor %}
</tr>
{% endfor %}

<a href="/">Home</a>
</tbody></table>
```

show_motion.html

```
1.      <h2>Historical Motion</h2>
2.
3.      <head>
4.          <title>Motion detected</title>
5.      </head>
6.
7.      <table border="1">
8.          <tbody><tr>
9.      <th>Date</th>
10.     <th>Time</th>
11.     <th>Zone</th>
12.     <th>Motion</th>
13.
14.
15.     </tr>
16.     {% for row in historical_motion %}
17.     <tr>
18.     {% for cell in row %}
19.
20.       <td> {{cell}} </td>
21.
22.     {% endfor %}
23.     </tr>
24.     {% endfor %}
25.
26.     <a href="/">Home</a>
27.     </tbody></table>
```

MÓDULS PRINCIPALS
servidor_web.py

```
1.      #!/usr/bin/python
2.      import sqlite3
3.      from flask import Flask, redirect, url_for,
  request
4.      from flask import render_template
5.      import time
6.      from datetime import datetime, date
7.
8.      app = Flask(__name__)
9.
10.     # Make connection with slqlite database.
11.     def connection(database):
12.         connect = sqlite3.connect(database)
13.         return connect
14.
15.     # Get data according the corresponding table.
16.     def get_data(table, elements):
17.         conn = connection('project.db')
18.         cursor = conn.execute("SELECT " +
  elements + " FROM " + table)
19.         group = [row for row in cursor]
20.         return group
21.
22.     # Get data desired by user.
23.     def get_data_selected(date1,date2):
24.         conn1 = connection('project.db')
25.         cursor1 = conn1.execute("SELECT
  tdate,ttime,zone,movimiento from movimientos
  WHERE tdate >= ? AND tdate <= ?",(date1,date2,))
26.         group1 = [row for row in cursor1]
27.         return group1
28.
29.     # The following four functions are the
  handling ones used to call the appropiate
30.     # resource, using the right method in each
  case.
31.     @app.route('/')
32.     def functionalities():
```

```
33.         return render_template('functionalities.h
  tml')
34.
35.     @app.route('/show_motion', methods=['GET', 'P
  OST'])
36.     def hist_motion():
37.         historical = get_data("movimientos", "tda
  te,ttime,zone,movimiento")
38.         return render_template('show_motion.html'
  , historical_motion=historical)
39.
40.     @app.route('/select_data', methods=['GET', 'P
  OST'])
41.     def select_data():
42.         global date1, date2
43.         if request.method == 'GET':
44.                 return render_template('select_da
  ta.html')
45.         elif request.method == 'POST':
46.             date1 = request.form.get('date1')
47.             date2 = request.form.get('date2')
48.
49.             return redirect(url_for('motion_desir
  ed'))
50.
51.     @app.route('/show_motion_desired', methods=['
  GET', 'POST'])
52.     def motion_desired():
53.         motion_desired = get_data_selected(date1,
  date2)
54.         return render_template('show_motion_desir
  ed.html', motion_desired = motion_desired)
55.
56.
57.     if __name__ == '__main__':
58.         app.debug = True
59.         app.run("192.168.1.19")
```

guardian.py

```
1. #!/usr/bin/env python
2. import ConfigParser
3. import sensors
4. from gmail import SendMail
5. import RPi.GPIO as GPIO
6. import time
7. from time import strftime, localtime
8. import sqlite3
9. import sys
10.     import os.path
11.
12.
13.     class Guardian(object):
14.         """Guardian is a security system for
  python"""
15.         # This funciton is the constructor of our
  class.
16.         def __init__(self):
17.             super(Guardian, self).__init__()
18.             self.binary_sensors = []
19.             self.sensors = {}
20.             self.pin_list = []
21.             self.options = []
22.             # Lists for alerts.
23.             self.lista = []
24.             self.timing_list = []
25.             self.diferencia_ultimo_primero = 0
26.
27.             self.config = self.check_cfg('configu
  ration.cfg')
28.             self.get_options(self.config)
29.             self.add_list_of_pins()
30.             # Save username,password and mode
  from the configfile.
31.             self.username = self.get_value(self.c
  onfig, 'Notifications', 'User email')
32.             self.password = self.get_value(self.c
  onfig, 'Notifications', 'Password')
33.             self.modo = self.get_value_int(self.c
  onfig, 'Notifications mode', 'Mode')
```

```
34.
35.             self.create_sensors()
36.
37.         # Add in binary_sensors list the sensor
  that the user would use.
38.         def create_sensors(self):
39.             self.binary_sensors.append(sensors.pi
  r.PIR("PIR",self.pin_list))
40.
41.         # Save in a file when de moviment was
  detected.
42.         def save_sensors(self):
43.             archivo = open('movimentos.txt','a')
44.             timex = strftime("%d-%m-%Y
  %H:%M:%S", localtime())
45.             archivo.write(timex+' '+'MOVIMIENTO
  DETECTADO\n')
46.             archivo.close()
47.             # In order to see in terminal.
48.             print timex + " MOVIMIENTO DETECTADO"
49.
50.         # Depending on how the user want to be
  notified, prepare the text for
51.         # the email message.
52.         def prepare_text_mode1(self,timex):
53.             msgbody = 'PELIGRO. MOVIMIENTO
  DETECTADO'
54.             text = timex + '    ' + msgbody
55.             return text
56.
57.         def prepare_text_mode2(self,mov,en,hace):
58.             timex = strftime("%d-%m-%Y
  %H:%M:%S", localtime())
59.             if mov == 1:
60.                 if en == 0:
61.                     msgbody = ('PELIGRO. {}
  MOVIMIENTO DETECTADO HACE {}
  SEGUNDOS').format(mov,hace)
62.                 else:
63.                     msgbody = ('PELIGRO. {}
  MOVIMIENTO DETECTADO EN {} SEGUNDOS HACE {}
  SEGUNDOS').format(mov,en,hace)
```

```
64.             else:
65.                 msgbody = ('PELIGRO. {}
  MOVIMIENTOS DETECTADOS EN {} SEGUNDOS HACE {}
  SEGUNDOS').format(mov,en,hace)
66.             text = timex + '    ' + (msgbody)
67.             return text
68.
69.         # Function which send an email with
  username and password configured in configuration
  file.
70.         def connect(self, contacts,text):
71.             subject ='INTRUSO'
72.             mailito = SendMail(self.username,self
  .password)          # Create an object of SendMail
  class.
73.             mailito.login()
74.
  # Call the member function login to gain acces.
75.             if os.path.isfile(contacts):
76.                 contactfile = open(contacts,'r')
                      # Open the file which contains
  the list of target email addresses.
77.                 contactlist = list()
                      # Create a list to store every
  contact separately.
78.                 for line in contactfile:
                      # Split every line by white
  spaces.
79.                     if line[1] == '\n':
                    # In order to delete the last
  line break.
80.                         line = line[:-1]
81.                     contactlist = contactlist+lin
  e.split(' ')
82.                 contactfile.close()
83.             else:
84.                 contactfile = open(contacts,'w')
85.                 print "File {} with recipients
  not found. Update the file mentioned writing all
  recipients separated by a white space in the same
  line".format(contacts)
86.                 sys.exit()
```

```
87.
88.             for contact in contactlist:
                    # For every contact in the list,
  send the email.
89.                 mailito.sendemail(contact,subject
  ,text)
90.
91.             mailito.closeSMTP()
                    # Call the member function
  closeSMTP to close smtp server.
92.
93.
94.         # Insert motion detected into a database.
95.         def insert_motion(self):
96.             conn = sqlite3.connect('project.db')
                     # Connect with database.
97.             zones = ["living room"]
98.             for zone in zones:
99.                 conn.execute("insert into
  movimientos (tdate,ttime,zone,movimiento) values
  (?, ?, ?, ?)",(strftime("%d-%m-%Y", localtime()),
100.                strftime("%H:%M:%S", localtime()),
101.                zone,'MOVIMIENTO DETECTADO'))
102.            print "Operation done successfully"
103.            conn.commit()
104.            conn.close()
105.
106.
107.        # Mode 1 of being notified.
108.        # If there are some moviment,save the
  information in a file and into a database
109.        # and finally send an email.
110.        def get_state(self):
111.            for sensor in self.binary_sensors:
112.                for pin in self.pin_list:
113.                    tiempo,valor = sensor.update_
  state(pin)
114.                    if tiempo != 0 and valor != 0
  :
115.                        self.save_sensors()
116.                        self.insert_motion()
```

```
117.                            text = self.prepare_text_
  mode1(tiempo)
118.                            self.connect('contact.txt
  ',text)
119.
120.        # Store hours, minutes and seconds in a
  different variables and convert all
121.        # into seconds.
122.        def conversion(self,timec):
123.            hh = int(timec[0:2])
124.            mm = int(timec[3:5])
125.            ss = int(timec[6:8])
126.            return hh, mm, ss
127.
128.        def segundos(self,hh,mm,ss):
129.            totalseconds = (hh*3600)+(mm*60)+ss
130.            return totalseconds
131.
132.        # Mode 2 of being notified.
133.        def get_state_2(self):
134.            for sensor in self.binary_sensors:
135.                for pin in self.pin_list:
136.                    tiempo,valor = sensor.update_
  state(pin)
137.                    # If there are no movement
  detected, and there are some message
138.                    # not notified, when 60
  seconds pass, we send an email.
139.                    if tiempo == 0 and valor == 0
  :
140.                        timing = strftime("%H:%M:
  %S", localtime())
141.                        hh,mm,ss = self.conversio
  n(timing)
142.                        totalseconds = self.segun
  dos(hh,mm,ss)
143.                        self.timing_list.append(t
  otalseconds)
144.                        long = len(self.timing_li
  st)
145.
```

```
146.                         if len(self.timing_list)>
  1 and len(self.lista)>=1:
147.                             diferencia = self.tim
  ing_list[long-1]-self.timing_list[0]
148.                             if diferencia > 60:
149.                                 print "Send for
  waiting too much"
150.                                 mov = len(self.li
  sta)
151.                                 text = self.prepa
  re_text_mode2(mov,self.diferencia_ultimo_primero,
  diferencia)
152.
153.                                 self.connect('con
  tact.txt',text)
154.                                 self.timing_list
  = []
155.                                 self.lista = []
156.
157.
158.     # If there are moviment detected, we check
  the difference between the two last successive
  movements and if these difference is less than 20
  seconds we wait for then send and email. If is
  bigger we send it.
159.                     if tiempo != 0 and valor != 0
  :
160.                         self.timing_list = []
161.                         hh,mm,ss = self.conversio
  n(tiempo)
162.                         totalseconds = self.segun
  dos(hh,mm,ss)
163.                         self.lista.append(totalse
  conds)
164.                         longitud = len(self.lista
  )
165.
166.                         if len(self.lista)>1:
167.                             diferencia_ultimo_pen
  ultimo = self.lista[longitud-1]-
  self.lista[longitud-2]
```

```
168.                                self.diferencia_ultim
  o_primero = self.lista[longitud-1]-self.lista[0]
169.                                diferencia_penultimo_
  primero = self.lista[longitud-2]-self.lista[0]
170.
171.    # If 40 seconds pass between movements that
  don't differ by more than 20 seconds,we send an
  email informing of gathered events.
172.                                if diferencia_ultimo_
  penultimo < 20:
173.                                    if self.diferenci
  a_ultimo_primero > 40:
174.                                        print "Sendin
  g gathered events"
175.                                        mov = len(sel
  f.lista)
176.                                        text = self.p
  repare_text_mode2(mov,self.diferencia_ultimo_prim
  ero,0)
177.                                        self.connect(
  'contact.txt',text)
178.                                        self.lista =
  []
179.                                else:
180.                                    mov = len(self.list
  a[:-1])
181.                                    text = self.prepare
  _text_mode2(mov,diferencia_penultimo_primero,dife
  rencia_ultimo_penultimo)
182.                                    self.connect('conta
  ct.txt',text)
183.                                    del self.lista[:-1]
184.
185.                            self.save_sensors()
186.                            self.insert_motion()
187.
188.        # Function to check if there is a file
  with given name and create new template or read
  given file.
189.        def check_cfg(self,file_name):
190.            config = self.crear_plantilla()
191.            if os.path.isfile(file_name):
```

```
192.              self.read_cfg(file_name, config)
193.          else:
194.              self.write_config(config,file_name)
195.              print ('Config file {} not found.
  Update the file mentioned '.format(file_name))
196.              sys.exit()
197.          return config
198.
199.      # Get value from a section and also for a
  specific option.
200.      def get_value(self,config, section, optio
  n):
201.          return config.get(section, option)
202.
203.      # Get integer value from a section and
  also for a specific option.
204.      def get_value_int(self,config, section, o
  ption):
205.          return config.getint(section,option)
206.
207.      # Function that give information about
  which field you have to fill
208.      def read_cfg(self,file_name, config):
209.          config.read(file_name)
210.          remaining = self.get_remaining(config
  )
211.          if len(remaining) != 0:
212.              for section, option in remaining:
213.                  print ('Fill option {} in
  section {}.Update file
  {}'.format(option, section, file_name))
214.              sys.exit()
215.
216.      # Check if in given configuration all
  values are filled, return a list with empty ones
217.      def get_remaining(self,config):
218.          remaining = []
219.          for k, v in config._sections.items():
220.              for k1, v1 in v.items():
221.                  if v1 == '':
222.                      remaining.append((k, k1))
223.          return remaining
```

```
224.    # Check in the file given how many options
  there are in order to know how many PIR sensors
  want the user use.
225.        def get_options(self,config):
226.            self.options = []
227.            for k, v in config._sections.items():
228.                for k1,v1 in v.items():
229.                    self.options.append(k1)
230.            return self.options
231.
232.     # With the options found, minus the
  username and the password, get the pin value from
  every PIR and add it in pin_list.
233.        def add_list_of_pins(self):
234.            # Minus 5 in order not to pick the
  username,password, neither the two comments and
  the mode of the configfile.
235.            for i in range(len(self.options[:-
  5])):
236.                pin = self.get_value_int(self.con
  fig,'GPIO BOARD', 'Pin' + ' ' + str(i+1))
237.                self.pin_list.append(pin)
238.            return self.pin_list
239.
240.        # Function that create a template to
  store configuration
241.        def crear_plantilla(self):
242.            # Allow no value = True in order not
  to see "None" next to the comments.
243.            config = ConfigParser.RawConfigParser
  (allow_no_value = True)
244.            config.add_section('GPIO BOARD')
245.            config.set('GPIO BOARD', 'Pin 1', '')
246.            config.set('GPIO BOARD', 'Pin 2', '')
247.            config.add_section('Notifications')
248.            config.set('Notifications', 'User
  email', '')
249.            config.set('Notifications', 'Password
  ', '')
250.            config.add_section('Notifications
  mode')
```

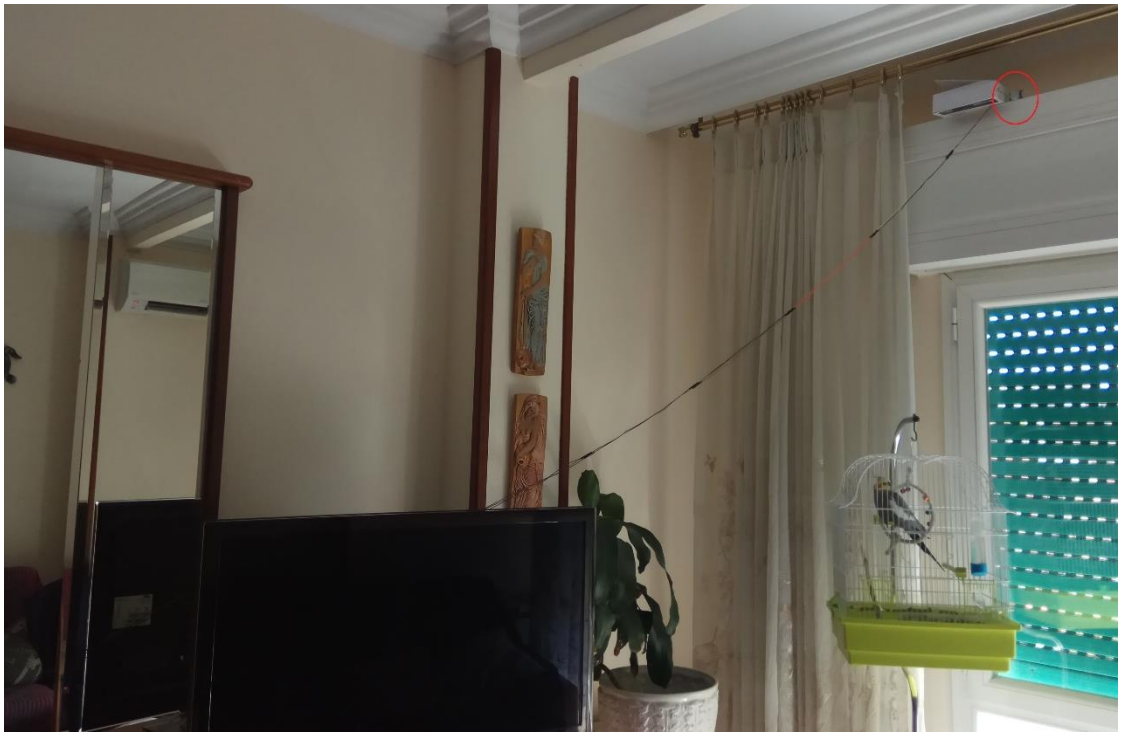
```
251.            config.set('Notifications mode', '#
  Mode 1: if you want to be notified every time')
252.            config.set('Notifications mode', '#
  Mode 2: if you want to be alerted with
  notifications ghatered')
253.            config.set('Notifications
  mode', 'Mode', '')
254.            return config
255.
256.        # Function that write in a given file the
  given configuration
257.        def write_config(self,config,file_name='c
  onfiguration.cfg'):
258.            with open(file_name, 'wb') as configf
  ile:
259.                config.write(configfile)
260.
261.     # Depending on the mode selected by user,the
  main function will execute the corresponding get_state
  unless the user make a keyboard interrupt.
262.        def main(self):
263.            try:
264.                if int(self.modo) == 1:
265.                    while True:
266.                        self.get_state()
267.                elif int(self.modo) == 2:
268.                    while True:
269.                        self.get_state_2()
270.                else:
271.                    print "Error with mode
  selected"
272.                    sys.exit()
273.            except KeyboardInterrupt:
274.                print "\nQuit"
275.                GPIO.cleanup()
276.
277.    # If this script is executed as a main
  program we will create an istance of the class
  and we will call the main function.
278.    if __name__=='__main__':
279.        guardian = Guardian()
280.        guardian.main()
```

10.2 Imatges del escenari utilitzat per les probes i col·locació dels sensors.

Il·lustració 23. Escenari escollit.

Il·lustració 24. Situació PIR que enfoca a la porta.

Il·lustració 25. Ampliació d'aquest darrer PIR.



Il·lustració 26. Situació PIR col·locat a la finestra.

Il·lustració 27. Ampliació d'aquest darrer PIR.